Segmentation of Motion Picture Images and Image Sequences

Peter Hillman

A thesis submitted for the degree of Doctor of Philosophy.
The University of Edinburgh.
December 20, 2002

Abstract

For Motion Picture Special Effects, it is often necessary to take a source image of an actor, segment the actor from the unwanted background, and then composite over a new background. The resultant image appears as if the actor was filmed in front of the new background. The standard approach requires the unwanted background to be a blue or green screen. While this technique is capable of handling areas where the foreground (the actor) blends into the background, the physical requirements present many practical problems.

This thesis investigates the possibility of segmenting images where the unwanted background is more varied. Standard segmentation techniques tend not to be effective, since motion picture images have extremely high resolution and high accuracy is required to make the result appear convincing. A set of novel algorithms which require minimal human interaction to initialise the processing is presented. These algorithms classify each pixel by comparing its colour to that of known background and foreground areas. They are shown to be effective where there is a sufficient distinction between the colours of the foreground and background.

A technique for assessing the quality of an image segmentation in order to compare these algorithms to alternative solutions is presented. Results are included which suggest that in most cases the novel algorithms have the best performance, and that they produce results more quickly than the alternative approaches.

Techniques for segmentation of moving images sequences are then presented. Results are included which show that only a few frames of the sequence need to be initialised by hand, as it is often possible to generate automatically the input required to initialise processing for the remaining frames. A novel algorithm which can produce acceptable results on image sequences where more conventional approaches fail or are too slow to be of use is presented.

Suggestions for future development of the techniques described in this thesis are included.

Declaration of originality

I hereby declare that the research recorded in this thesis and the thesis itself was composed and originated entirely by myself in the Department of Electronics and Electrical Engineering at The University of Edinburgh.

Peter Hillman

Acknowledgements

Firstly, I would like to thank my supervisors, Drs John Hannah and David Renshaw, for their support, guidance and ideas.

Many thanks to other people who have helped me acquire knowledge — Dr Dean McNeill for his help with classifiers, Robert Thomson for his help with Genetic Algorithms and Dr Andrew Peacock for his help with almost everything else, most particularly on statistics and Bayes' theory. Thanks also to Andrew for his work on the Vision Systems code library, which saved much tedious programming.

Thanks also go to David Stewart of the departmental computer support, for his friendliness and tolerance despite the processing and data storage requirements that work for this thesis involved.

Thanks to Tim Alexander and Steve Sullivan of Industrial Light and Magic, to David Scott of the Computer Film Company, and Paddy Eason of the Moving Picture Company for their help from the world of Motion Picture Post Production.

A big thank-you to Gema for allowing me to abuse so extensively such an unflattering picture of her. *Gracias por todo.*

This thesis has been supported by EPSRC under studentship award number 99303086.

No Teddybears were harmed during the production of this thesis

Contents

Declaration of originality . . . iii
Acknowledgements . . . iv
Contents . . . v
List of figures . . . ix
List of tables . . . xii
List of algorithms . . . xiii
Acronyms and abbreviations . . . xiv
Nomenclature . . . xv

1 Introduction **1**
1.1 Background . . . 1
1.2 Contributions . . . 2
1.3 Aims . . . 2
1.4 Structure . . . 2
1.5 Summary . . . 4

2 Imaging **5**
2.1 Introduction . . . 5
2.2 Digital Images . . . 5
2.2.1 Pixels and Channels . . . 6
2.2.2 Bit Depth . . . 6
2.2.3 Digital Image Storage . . . 7
2.3 Digital Compositing . . . 7
2.3.1 Alpha Channels . . . 8
2.3.2 Clean Foregrounds . . . 9
2.4 Representing and Processing Colour . . . 10
2.4.1 RGB Colour Space . . . 11
2.4.2 Hue, Saturation, Intensity (HSI) Colour Space . . . 12
2.4.3 $CIE - Lab$ Colour Space . . . 14
2.5 Motion Picture Imaging . . . 16
2.5.1 Imaging on film . . . 16
2.5.2 Digital Imaging . . . 17
2.5.3 Scanning . . . 18
2.5.4 Motion Picture Resolutions . . . 18
2.5.5 Is Film Dying? . . . 19
2.6 Summary . . . 21

3 Still Colour Image Segmentation **22**
3.1 Introduction . . . 22
3.2 Motion Picture Segmentation Techniques . . . 22
3.2.1 Lumakey . . . 23
3.2.2 Differencing . . . 24

3.2.3 Chromakey . . . 24
3.2.4 Depth Perception . . . 26
3.2.5 Rotosplining . . . 27
3.3 Pixel-based Segmentation . . . 28
3.4 Region Growing . . . 28
3.5 Snakes and Edges . . . 31
3.5.1 Intelligent Scissors . . . 33
3.6 Textures . . . 33
3.7 Summary . . . 37

4 Moving Image Segmentation **39**
4.1 Introduction . . . 39
4.2 Difference Matting . . . 39
4.2.1 Differencing with Moving Backgrounds . . . 41
4.3 Block Matching . . . 41
4.4 Optical Flow . . . 43
4.5 Motion Estimation . . . 44
4.6 Active Contours . . . 45
4.7 Motion-based Region Tracking . . . 46
4.8 Summary . . . 47

5 Alpha Channel Estimation for Single Frames **48**
5.1 Introduction . . . 48
5.2 The Need for Human Interaction . . . 48
5.2.1 Creating a *Hint Image* . . . 50
5.3 Alpha Estimation by Inverting the Compositing Equation . . . 50
5.4 Assessment of algorithms . . . 54
5.5 Nearest Pixel Estimation . . . 54
5.6 Colour Correction . . . 55
5.7 Cluster-based Segmentation . . . 55
5.7.1 The Stripe Method of Selecting Pixels . . . 58
5.7.2 Alpha range adjustment . . . 60
5.8 Selective clean colour estimation . . . 62
5.9 Nearest Colour estimation . . . 63
5.10 Adaption with Principal Components Analysis . . . 65
5.11 Multiple Classification . . . 68
5.12 Improvements using Region Growing . . . 68
5.12.1 Region Growing to Expand the Known Areas . . . 71
5.12.2 Region Growing for Alpha Estimation . . . 71
5.13 Backlighting . . . 74
5.13.1 Estimating the Clean Backlight Colour . . . 79
5.14 Summary . . . 80

6 Comparison with Alternative Techniques **82**
6.1 Introduction . . . 82
6.2 The *KnockOut* Algorithm . . . 83
6.3 Ruzon and Tomasi's Technique . . . 84

6.4 The Technique of Chuang *et al* . . . 86
6.5 A Genetic Algorithm Solution to Bayesian Matting . . . 88
6.5.1 Modelling Spatial Coherency Using $L(\alpha)$ . . . 88
6.5.2 Optimisation of the Bayesian problem . . . 90
6.5.3 Genetic Algorithms . . . 90
6.6 Results . . . 93
6.6.1 Quantitive Results . . . 93
6.6.2 Bluescreen Results . . . 94
6.6.3 The *Edith* Image . . . 105
6.6.4 The *Teddy* Image . . . 113
6.6.5 The *Gema* Image . . . 120
6.7 Summary . . . 127

7 Adaptation of Alpha Estimation to Image Sequences 129
7.1 Introduction . . . 129
7.2 Hint Image Estimation vs. Alpha Channel Estimation . . . 130
7.3 Processing the Unknown Area . . . 131
7.4 Hint Images from Motion Fields . . . 132
7.4.1 Calculating the Motion Field . . . 134
7.5 Problems with Motion Vector Field Segmentation . . . 134
7.5.1 Large Movements and Deformations . . . 134
7.5.2 Computational Requirements . . . 135
7.5.3 Occlusions and Disocclusions . . . 135
7.6 Colour-based Comparison . . . 136
7.6.1 Simple Colour Distance Classification . . . 136
7.6.2 Increasing the Speed of the Algorithm . . . 137
7.6.3 Colour Distance–to–Means Classification . . . 144
7.6.4 Colour Probability Classification . . . 145
7.6.5 Summary . . . 147
7.7 Noise Removal . . . 147
7.7.1 Small Region Removal . . . 148
7.7.2 Region Voting . . . 152
7.7.3 Dilation . . . 154
7.8 Progressive and Non–Progressive Estimation . . . 154
7.9 Multiple Reference Frames . . . 155
7.10 Alpha Channel Estimation in Motion Picture Sequences . . . 156
7.11 Results . . . 156
7.11.1 The *Dragonheart* Sequence . . . 158
7.11.2 The *Teddy* Sequence . . . 168
7.12 Summary . . . 172

8 Conclusions 173
8.1 Introduction . . . 173
8.2 Summary . . . 173
8.2.1 Performance . . . 174
8.3 Parameter Selection . . . 176
8.3.1 Colourspace Selection . . . 176

8.4 Implementation Considerations . . . 177
8.4.1 User Interfaces . . . 177
8.4.2 Parallel Implementation . . . 177
8.5 Critical Discussion . . . 178
8.6 Recommendations for Future Work . . . 178
8.6.1 Quantitive and Qualitive results . . . 178
8.6.2 Non Colour–based Segmentation . . . 179
8.6.3 Combination of Algorithms . . . 179
8.6.4 Extended Clean Backgrounds . . . 180
8.6.5 Hint Image Initialisation . . . 181
8.7 Concluding Remarks . . . 182

References **183**

A Test Data **192**

B Publication **201**

C Webpage for evaluation **202**

D Example code **203**
D.1 Coding Style . . . 203
D.1.1 Program Size . . . 203
D.2 The Vision Systems Group Library . . . 203
D.3 Sample Code . . . 204
D.3.1 VS_floatpixel Class . . . 204
D.3.2 Compose Program . . . 206

List of figures

2.1 Diagram of a digital image 6
2.2 Compositing requires a clean foreground image 10
2.3 Plot of part of the RGB colour space 12
2.4 Plot of part of the HSI colour space 13
2.5 The *Lena* image quantised in RGB and CIE-Lab space 14
2.6 3d plot of part of the $CIE - Lab$ colour space 15
2.7 Bayer pattern of colour sensors 17

3.1 Example of alpha channel generation using Lumakey 23
3.2 Example of alpha generation using differencing 24
3.3 Example of alpha generation using CFC's *Keylight* Chromakey algorithm . . . 25
3.4 Linear snake fitted to the boundary of the *Gema* image 32
3.5 Histogram based texture segmentation 34

5.1 Curves used for input to Corel *KnockOut* 49
5.2 An example Hint Image 51
5.3 Calculating α from clean colours 51
5.4 A 50% grey pixel could have any alpha value 52
5.5 Calculating α when the clean colours are estimates 53
5.6 Location of pixels used as clean colours in nearest pixel estimation 55
5.7 Results of nearest pixel estimation technique 56
5.8 Scheme for correcting estimated foreground colour 57
5.9 Location of pixels used for clusters in the simple cluster based algorithm 57
5.10 Results of using the cluster–based algorithm on the *Gema* test image 59
5.11 Problem caused by selection of pixels using algorithm of section 5.7 60
5.12 The *stripe method* of selecting pixels 60
5.13 Results of using the *stripe* algorithm on the *Gema* test image 61
5.14 Problem caused by high variance clusters 62
5.15 Results of using the Nearest Colours algorithm on the *Gema* test image 64
5.16 Cluster of pixel colours from *Gema* image, showing prolate shape 65
5.17 Cluster of points in RGB space with the line $\overline{\mathbf{p_0 p_1}}$ 66
5.18 Position of points used to classify in colour space 67
5.19 Results of using the Principal Axis algorithm on the *Gema* test image 69
5.20 Multi-classification algorithm on the *Gema* test image 70
5.21 Region growing algorithm on the *Gema* test image 75
5.22 Poor results occur with backlit foregrounds 76
5.23 Graph of pixel values with backlighting 76
5.24 Position of points used to classify in the presence of backlighting 77
5.25 Result of using backlight alpha estimation 79

6.1 Different methods for selecting clusters of foreground and background pixels . 83
6.2 Model used by Ruzon and Tomasi 85

6.3 Treating an image as a graph . . . 89
6.4 Ground truth alpha channel for *Rachael* test image . . . 93
6.5 CFC's *Keylight* on the *Rachael* test image . . . 96
6.6 Corel *KnockOut* on the *Rachael* test image . . . 97
6.7 Adobe *Photoshop* on the *Rachael* test image . . . 98
6.8 Ruzon and Tomasi's algorithm on the *Rachael* test image . . . 99
6.9 Chuang *et al* on the *Rachael* test image . . . 100
6.10 Principal Axis algorithm without alpha correction on the *Rachael* test image . . . 101
6.11 Principal Axis algorithm with alpha correction on the *Rachael* test image . . . 102
6.12 Genetic Algorithm on the Rachael test image . . . 103
6.13 Adobe *Photoshop* on the *Edith* image . . . 106
6.14 Corel *KnockOut* on the *Edith* image . . . 107
6.15 Ruzon and Tomasi's algorithm on the *Edith* image . . . 108
6.16 Principal Axis algorithm on the *Edith* image . . . 109
6.17 Genetic algorithm on the *Edith* image . . . 110
6.18 Details of alpha channels produced from composited *Edith* image . . . 112
6.19 Adobe *Photoshop* on the *Teddy* image . . . 114
6.20 Corel *KnockOut* on the *Teddy* image . . . 115
6.21 Ruzon and Tomasi's algorithm on the *Teddy* image . . . 116
6.22 Chuang *et al* on the *Teddy* image . . . 117
6.23 Principal Axis algorithm on the *Teddy* image . . . 118
6.24 Genetic algorithm on the *Teddy* image . . . 119
6.25 Adobe *Photoshop* on the *Gema* image . . . 121
6.26 Corel *KnockOut* on the *Gema* image . . . 122
6.27 Results of running Ruzon and Tomasi's algorithm on the *Gema* image . . . 123
6.28 Chuang *et al* on the *Gema* image . . . 124
6.29 Principal Axis algorithm on the *Gema* image . . . 125
6.30 Genetic algorithm on the *Gema* image . . . 126
6.31 Chart of RMSE error of each algorithm combined for both test images . . . 127

7.1 The unknown area is not always consistent between frames . . . 131
7.2 Motion vector field between frames 3 and 2 of the *Dragonheart* sequence . . . 132
7.3 Hint image generated using the motion field of Fig. 7.2 . . . 133
7.4 Distribution of colour in a 256×256 block of an image . . . 138
7.5 Distribution estimated as a collection of boxes . . . 139
7.6 Selecting squares to represent the search aperture . . . 141
7.7 Hint image created by the bounding set algorithm . . . 143
7.8 A cluster of points approximated as four means . . . 144
7.9 Hint image created by the distance-to-means algorithm . . . 145
7.10 Hint image created by the probability–based algorithm . . . 146
7.11 Classes of pixels in part of frame 1 of *Teddy* sequence . . . 148
7.12 Max-tree noise removal in a sample image . . . 151
7.13 Fig. 7.10 after applying small region removal . . . 152
7.14 Frame 4 of the *Dragonheart* sequence partitioned into regions . . . 153
7.15 Fig. 7.10 after applying region voting . . . 153
7.16 Complete system for alpha channel estimation of motion picture sequences . . . 157
7.17 Results for frame 1 of the *Dragonheart* sequence . . . 159

7.18 Hint images produced to process frame 21 of the *Dragonheart* sequence 160
7.19 Results for frame 21 of the *Dragonheart* sequence 161
7.20 Hint images produced to process frame 5 of the *Dragonheart* sequence 162
7.21 Results for frame 5 of the *Dragonheart* sequence 163
7.22 Hint images produced for frame 10 of the *Dragonheart* sequence 164
7.23 Results for frame 10 of the *Dragonheart* sequence 165
7.24 Hint images produced to process frame 15 of the *Dragonheart* sequence 166
7.25 Results for frame 15 of the *Dragonheart* sequence 167
7.26 Hint images produced to process frame 2 of the *Teddy* sequence 168
7.27 Results for frame 2 of the *Teddy* sequence 169
7.28 Hint images produced to process frame 3 of the *Teddy* sequence 170
7.29 Results for frame 3 of the *Teddy* sequence 171

8.1 Extended clean background generated from the *Dragonheart* sequence 180
8.2 Attempt at automatic hint generation 181

A.1 The *Rachael* test image 194
A.2 The *Gema* test image 195
A.3 The *Teddy* test image - frame 1 of the *Teddy* Sequence 196
A.4 Frames 2 and 3 of the *Teddy* Sequence 197
A.5 The *Edith* test image 198
A.6 Frame 15 of the *Dragonheart* sequence 199
A.7 The *Dragonheart* sequence 200

List of tables

2.1 Comparison of different image data formats 19
2.2 Comparison between digitised film images and the Sony HDW-F900 Camcorder 20

5.1 Relative performance of the algorithms presented in Chapter 5 81

6.1 Parameters used for Principal Axis algorithm on the *Rachael* test image 95
6.2 RMSE performance of algorithms on the *Rachael* test image 104
6.3 RMSE accuracy of alpha channels produced from composited *Edith* image . . . 111
6.4 Run times of unoptimised implementations of algorithms on *Teddy* image . . . 113

A.1 Details of test images 193

List of algorithms

1	Binary compositing	8
2	Simple region growing	29
3	Method to form clusters of foreground of background points	58
4	Alpha range adjustment	62
5	Nearest Colours algorithm to select clean foreground colours	63
6	Region growing approach to alpha estimation	72
7	Alpha estimation with the Region Growing approach	73
8	Genetic Algorithm to solve Bayesian Matting	92
9	Using a motion field to generate a hint image	133
10	Simple colour-based hint image generation	137
11	Calculating a bounding set for a cluster	140
12	Building a hint image using recursive classification and bounding sets	142
13	Small region removal algorithm	149
14	Creating a Region Image	149
15	Creating a Region Adjacency Graph	150
16	Building a Max-tree using a Region Adjacency Graph	150
17	Colour-distance based classification with multiple keyframes	155

Acronyms and abbreviations

AM	Amplitude Modulation
BBC	British Broadcasting Corporation
bpc	bits per channel
bpp	bits per pixel
CCD	Charge Coupled Device (image sensing technology)
CFC	Computer Film Company
CIE	Commission Internationale de l'Eclairage (International Commission on Illumination) — international standardisation body which defines colour spaces
CMOS	Complementary Metal Oxide Semiconductor (digital logic and alternative image sensing technology to CCD)
DCT	Discrete Cosine Transform
FM	Frequency Modulation
fps	frames per second
GA	Genetic Algorithm
GMM	Gaussian Mixture Model
HSI	Hue, Saturation Intensity (colourspace)
HSV	Hue, Saturation Value (colourspace)
KNN	K Nearest Neighbour (classifier)
PCA	Principal Components Analysis
PCD	Kodak PhotoCD (image format)
RGB	Red, Green, Blue (colourspace)
RMSE	Root Mean Squared Error
SAD	Summed Absolute Difference

Nomenclature

α	alpha channel, or alpha value for a pixel
B	clean background image
$\boldsymbol{b}$	vector triple representing colour of background pixel
$\boldsymbol{b'}$	corrected background colour
C	cluster of foreground or background pixels near to point to be classified
E	matrix of eigenvectors
$\boldsymbol{e}$	eigenvector
F	clean foreground image
$\boldsymbol{f}$	vector triple representing colour of foreground pixel
$\boldsymbol{f'}$	corrected foreground colour
λ	eigenvalue
$\boldsymbol{\mu}$	mean colour of a cluster or subcluster of pixels
$\boldsymbol{m}$	vector triple representing backlight colour
$\boldsymbol{n}$	normal to a plane
P	composite image or input image
$\boldsymbol{p}$	vector triple representing colour of pixel in composite (output) or input image
$\boldsymbol{p_0}, \boldsymbol{p_1}$	end points of principal axis of cluster in colourspace
$\boldsymbol{q}$	point which is the nearest point on the line $\overrightarrow{\boldsymbol{fb}}$ to $\boldsymbol{p}$ in colourspace
$\overline{R}, \Sigma$	3×3 correlation matrix
$\boldsymbol{s}$	a vector triple representing colour of pixel in input image under classification
U	set of unknown pixels to be classified

Chapter 1
Introduction

1.1 Background

Since the advent of motion picture imaging, film makers have experimented with optical special effects — the creation or modification of cinematic images to create the illusion of a scene which did not really exist. In particular, different sequences could be combined together to produce a new image. Thus, Superman could be made to fly by combining a shot of an actor in a studio with a shot of a cityscape. The images were combined, or *composited*, by printing each sequence onto the same piece of film. In order that only the required parts of each film are included in the final image, two complementary rolls of film — called *mattes* — had to be created. This film was translucent where the image must be removed and transparent where it was required. Mattes could be created using optical processing, but this relied on very stringently controlled environments when filming the individual sequences. If this was not possible, the mattes had to be painted onto sheets of glass and then photographed.

With the introduction of digital image processing, compositing images became a digital process. Optical matte creation has been replaced by digital processing and painting by hand by digital painting packages.

Even with these advances in image compositing, the stringent requirements for filming the individual sequences have been relaxed only slightly. The favoured way of creating a sequence of an actor so that it can be superimposed on a different background is to film the actor in front of a blue screen, as shown in Fig. A.1. If the background is more varied, it is usually necessary to revert to hand–generated matte painting packages.

This thesis investigates possible algorithms to generate such matte channels from motion picture resolution image sequences where the background is less restrictive. Algorithms are proposed which are capable of producing mattes from images with varied and natural backgrounds.

1.2 Contributions

The contribution of this thesis is to present a group of segmentation algorithms. These algorithms can generate mattes from motion picture resolution image sequences. The results they produce are generally superior to existing algorithms, and run faster than those systems. An adaptation to these algorithms is proposed to handle the case where the foreground is illuminated from behind.

1.3 Aims

The "Holy Grail" of motion picture segmentation algorithms would be to take any arbitrary image sequence and to segment the background from the foreground. Since even the human vision system is incapable of separating people from their backgrounds in some circumstances, this is clearly an extremely difficult, if not impossible, task. Even though the algorithms presented here perform reasonably well on images with natural backgrounds, the intention is primarily to allow automatic segmentation of a wider range of images. Ideally, it should be possible to segment images that may were not originally intended to be segmented, such as archive film or film which needs to be segmented to disguise some error which occurred during filming.

1.4 Structure

The structure of this thesis is as follows:

- Chapter 2 of this thesis introduces the process of digital compositing, and alpha channels, the digital equivalent of the optical matte. It then examines the nature of motion picture film images (in terms of their resolution and origin) and the process of converting images to a digital format to allow processing. The new high resolution digital motion picture format is also discussed.

  Since colour processing is at the heart of the novel algorithms presented here, an examination of different colour spaces is presented.

- Chapter 3 gives an overview of segmentation of still images. There are a huge variety of algorithms and techniques used to segment still images. A small number of these are

currently used by the Motion Picture industry. This chapter gives a brief survey of such techniques.

- Chapter 4 is an overview of segmentation of moving images. Again, the chapter is a brief survey of different techniques available to segment images.

- Chapter 5 presents novel algorithms that can produce mattes from static motion picture resolution images. The human interaction required is described first, followed by a set of simple algorithms. These algorithms motivate the more complex algorithm presented at the end of the chapter.

- Chapter 6 examines other matte creation algorithms, all of which were published while the algorithms of chapter 5 were being developed. Such algorithms have motivated a new approach using a Genetic Algorithm, which is presented in this chapter.

  Chapter 6 also presents a selection of qualitative and quantitive results, comparing existing algorithms and systems to the novel algorithms presented in this thesis.

- Chapter 7 presents a set of algorithms to adapt the algorithms of chapters 5 and 6 to moving sequences. The human input required for still images need not be supplied for every frame of a moving sequence, since this algorithm is capable of producing the input for some subsequent frames automatically. Results of applying this algorithm to high resolution sequences are presented.

- Chapter 8 presents conclusions and a summary of the thesis, a discussion of its limitations, and suggestions of areas for future research and development.

- Standard test images commonly used to test image processing algorithms are not appropriate for this work, since they do not possess sufficient resolution. It was therefore necessary to obtain of create a suitable set of images in order to test the algorithms. These images are described and reproduced in Appendix A

- Appendix B lists publications arising directly from this work.

- Appendix C reproduces a webpage used in attempt to obtain a qualitive assessment of the quality of the results presented in this thesis

- Appendix D describes the programs written to implement the algorithms presented in this thesis.

1.5 Summary

The theme of this thesis is the segmentation of motion picture resolution images and image sequences in order that they may be used in digital image compositing. Currently, an image that is to be segmented by an accurate automatic algorithm for use in compositing must be filmed in a very stringent conditions (typically against a carefully lit blue screen). With use of algorithms presented in this thesis, it is possible to use less restrictive backgrounds.

Chapter 2
Imaging

2.1 Introduction

Compositing is the process of overlaying different parts of images together. Different *elements* are filmed separately and then combined together to form a unified image. Since it is rarely possible to film the element in isolation with no unwanted objects (such as the background behind the object) appearing in the image as well, a *matte* or *alpha channel* must be created in order to indicate which parts of the image are to be included in the final image and which are to be discarded. Any traces of unwanted objects must be removed, and a *clean foreground* image produced. Creating such a matte and a clean foreground image is a special case of image segmentation. Such a segmentation algorithm must operate on motion picture resolution images, which are extremely high in resolution, and have unusual noise characteristics.

The structure of this chapter is as follows: Section 2.2 considers the nature of a digital image. Section 2.3 defines the process of digital compositing and introduces the two outputs of a matte producing segmentation algorithm, the *alpha channel* and the *clean foreground image*. Work in this thesis involves motion picture colour images. Techniques for representing and processing colour are considered in section 2.4. Section 2.5 examines the nature of motion picture imaging, and conversion of film images to digital formats. The chapter is summarised in Section 2.6.

2.2 Digital Images

A digital image is a numerical representation of an image. The image is split into individual *picture elements*, *pels* or *pixels*, each of which has a fixed size and shape. An image is formed out of a rectangular array of these pixels.

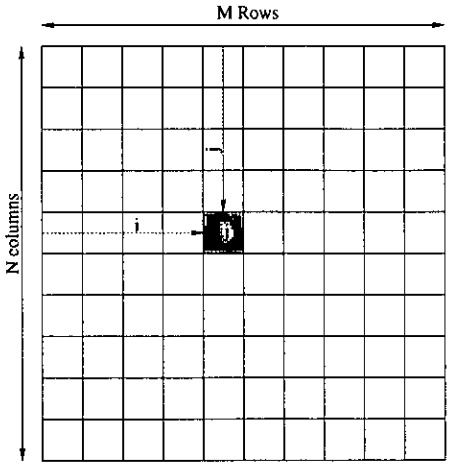


Figure 2.1: *Diagram of a digital image, showing the position of a pixel at ij*

2.2.1 Pixels and Channels

Fig. 2.1 shows a representation of an image. The $M \times N$ image has M rows and N columns. In this thesis, a pixel in an image I will be referred to as I_{ij}, as pixel ij, or as pixel (i, j) interchangeably. In all cases, the pixel is in the i^{th} column and the j^{th} row, as shown in Fig. 2.1.

The information stored at the pixel depends on the nature of the image. An image contains at least one *channel* of information. In a grey image, there is only one channel. Only one value is required for each pixel. A colour image typically requires three channels, *i.e.* three values per pixel, as explained in Section 2.4. In remote sensing applications, such as satellite and radar imaging, an image can contain any number of channels.

2.2.2 Bit Depth

Bit depth refers to the precision with which each channel of each pixel is represented. Using 1 bit per pixel (*bpp*) in a single channel image produces a binary black and white image. For colour photographic images, 8 bits are typically used for each channel (8 *bpc*), providing 256 possible values. As there are three channels, there are 24 bits per pixel and 16,777,216 possible colours for each pixel. This is generally considered to be adequate to ensure that the quantisation effects caused by finite precision are not noticeable. However, processing the image (for example increasing the contrast of the image) may make quantisation artifacts visible. Using more memory to represent images with higher bit depths is then worthwhile.

2.2.3 Digital Image Storage

The manner in which images are stored depends on the application. A simple way of storing an image is in a linear fashion. Each row of the image is stored consecutively. Thus, the C expression

```
Image[ (j*M+i)*3 + n ]
```
(2.1)

would be used to access channel n of pixel ij in a three channel image of width M stored at address `Image`. It is also common to store the columns consecutively (rather than the rows) and to store each channel separately (rather than interleaved).

Certain image file formats store images in a compressed format. It is possible to remove information that is not detectable to the eye (*lossy compression*) or to find redundancies in the data (such as many pixels in an area all with the same colour) and thus to find a more efficient representation of the image (*lossless compression*).

2.3 Digital Compositing

Digital compositing is defined by Brinkmann [1] as

> The digitally manipulated combination of at least two source images to produce an integrated result

Typically, just two images are involved, one being the background and the other some object that is to appear in front of it, which will be referred to as the foreground. However, some sequences in the film *Titanic* [2] involve several hundred source images combined to form the final result [1].

An image that is part of a composite image is called an *element*. This thesis is concerned with the generation of elements from source images.

2.3.1 Alpha Channels

The source image will contain some parts that are not required in the element. These unwanted parts must not be included in the composite image. An *alpha channel* indicates which pixels are part of the element and which are not. This alpha channel can be stored as a fourth channel of the image.

Suppose it is required to combine two images B and F together to produce a final result P. The compositing process is simply defined in the *binary compositing algorithm* (Algorithm 1)

```
for each pixel ij:
   if pixel ij is part of element F then
      set P_ij = F_ij
   else
      set P_ij = B_ij
   endif
next pixel
```

Algorithm 1: *Binary compositing*

In order to decide whether a pixel is part of F or B, an *alpha channel*, *matte* or *mask* can be used: This can be a binary image M: If pixel ij is white in mask M, then set $P_{ij} = F_{ij}$, else set $P_{ij} = B_{ij}$. However, this binary decision is often not accurate enough: Some pixels in the final image P will need to be a mixture of both elements F and B. The main reason that this could occur is *Motion Blur*. Devices that capture images do so by integrating light over a period of time. It may be the case that the object in element F is moving with respect to the background B. If the image was not a composite, but a genuine shot of the object moving over the background, some pixels in the image would be a mixture of F and B, because for part of the exposure they would be part of F, and for the rest of the time they would be part of B. In order to recreate this effect, a greyscale alpha channel is used rather than a binary one, where each pixel in M takes a value α in the range $[0, 1]$. α_{ij} can be considered as the fraction of the exposure time that pixel P_{ij} is to appear to be part of element F. Another reason that pixels may be a mixture of more than one element is sub-pixel aliasing: a single pixel may be simultaneously illuminated by both background and foreground, because it is not infinitely small, or because the focus is not perfectly sharp.

To compose an element with a greyscale alpha channel over another object, the full *Alpha Compositing Equation* [1, 3, 4, 5, 6] is used:

$$P_{ij} = \alpha_{ij} F_{ij} + (1 - \alpha_{ij}) B_{ij} \tag{2.2}$$

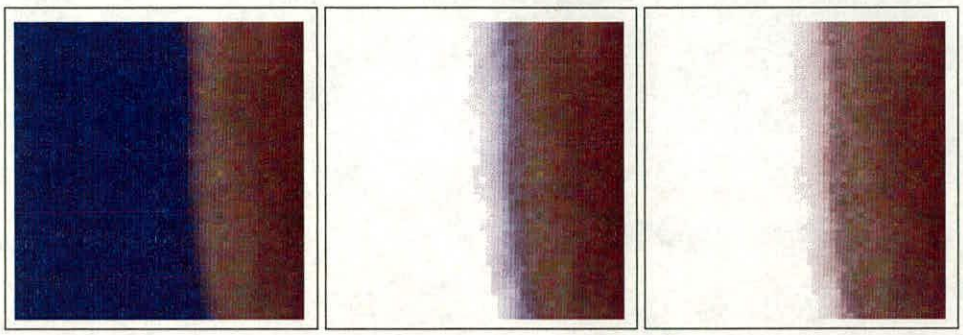
There are conditions where combining two objects together using an alpha channel will not produce an accurate enough result: some objects may transmit different colours of light differently, or may reflect or refract light from other objects. A pair of tinted spectacles is an example of an object that cannot be composited using an Alpha Channel. Zongker *et al* [7] developed a system that generates an "environment matte" that is capable of correctly reproducing transmission, reflection and refraction of an object. The object is placed in front of monitors, which display a sequence of known patterns, and the object is photographed in front of each one. A full description of the way the object interacts optically with its surroundings can therefore be deduced. This technique cannot be applied to moving objects (unless their motion can be very accurately controlled) because the object must not move while it is photographed multiple times.

However, alpha channels are a standard tool for imaging work. They are produced by raytracing packages such as Pixar's *RenderMan* [8, 9] (which produce computer-generated images of elements along with an alpha channel that indicates the percentage coverage of each pixel in the image by the element). Compositing packages such as Puffin Design's *Commotion* [10] use alpha channels for simple compositing as well as for other "Matte Algebra" [6] in order to combine elements together. Alpha channels are also used by Adobe's Photoshop [11] to create shadows and glows around objects [12]. Since alpha channels are commonly used for many purposes, tools that can create them are valuable.

Where an element is generated artificially (for example by a rendering package), the accompanying alpha channel is usually generated automatically. However, generating elements from a photographed scene requires the alpha channel for each required element in the scene to be produced separately. The alpha channel will cause all parts of the scene not required in the final element to disappear.

2.3.2 Clean Foregrounds

As well as generating an alpha channel, it is also necessary to generate a modified version of the foreground image. Fig. 2.2 illustrates this. Some pixels in the source image (from which



(a) Original image

(b) Composite formed with original image

(c) Composite formed with clean foreground image

Figure 2.2: *Bluescreen-originated element (a) composited over a white background using the original image (b) and a clean foreground (c). Parts of the bluescreen are visible in the composite made with the original image (b). The composite appears correct when using a clean foreground (c)*

the element is to be extracted) will be a combination of required foreground, and unwanted background. These pixels should be assigned an intermediate alpha value to reflect this. So, if in the source image the required foreground is red and the unwanted background blue, and a pixel is a 50% mix of background and foreground, the final colour of the pixel will be some shade of magenta and the alpha value should be 0.5. However, if this magenta pixel is incorporated into the final image P, there will be a noticeable blue fringe around the element, because parts of the background will have been transferred. The final pixel in image P will be 50% new background, 25% foreground, and 25% unwanted background. This is incorrect - it should be 50% element and 50% new background.

Therefore a second step in the extraction of an element is the removal of traces of background colour from those pixels that do not entirely belong to the foreground. In this thesis, the resultant image will be called a *clean foreground* image.

2.4 Representing and Processing Colour

The human eye is sensitive to red, green and blue light, and therefore most image sensors that produce colour images which are to be viewed by a human eye are also sensitive to red, green and blue light. Every pixel in a full colour image will thus have at least three channels of data.

When colour images are processed, a decision must be taken as to how to process these three channels. Some common approaches are:

Average the channels together to produce a greyscale image. This technique is not really colour image processing, but is a useful technique used to apply a greyscale image processing technique to a colour image.

Treat each channel separately. Each separate channel can be considered to be a greyscale image. Once each channel is processed, the channel producing the best result can be used, or alternatively the results from the three separate channels can be fused together.

Treat colours as vectors and form a *Colour Space*. Thus a pixel with channel values k, l and m is considered to be a point in colour space with co-ordinates $[klm]^T$. The difference in colour between two pixels with colour vectors $\mathbf{s}$ and $\mathbf{t}$ can be found simply by measuring the Euclidean distance $|\mathbf{s}-\mathbf{t}|$ using the between the points in colourspace. This "distance between colours" must not be confused with the "distance between pixels" which is the separation of the two points in the image.

If a colour space is used, some consideration of the nature of that space is important. Colour spaces are considered in detail by Skarbek [13] and by Hunt [14]. Ford and Roberts [15] provide formulae for conversions between many different colour spaces. The colour spaces detailed below represent many features common among all colour spaces.

2.4.1 RGB Colour Space

The simplest colour space is RGB space. Here, the co-ordinates (r, g, b) of a pixel in colour space are simply the values of the red, green and blue channels respectively. It is useful mainly because image sensors usually produce rgb channel data directly so no conversion need take place, saving time and preserving accuracy. Since information between separate channels is not merged, noise originating in the sensor present in each channel is uncorrelated. However, RGB space presents several problems for image processing. For example, it may be that two different shades of red are further apart in colour space than the colours of a red object and a green object. Thus, two different parts of the same object may be treated as being different objects, but two completely different objects would be treated as one object. A slight modification to RGB space is the 2d *normalised RGB space*, used by Healey [16] to perform image segmentation.

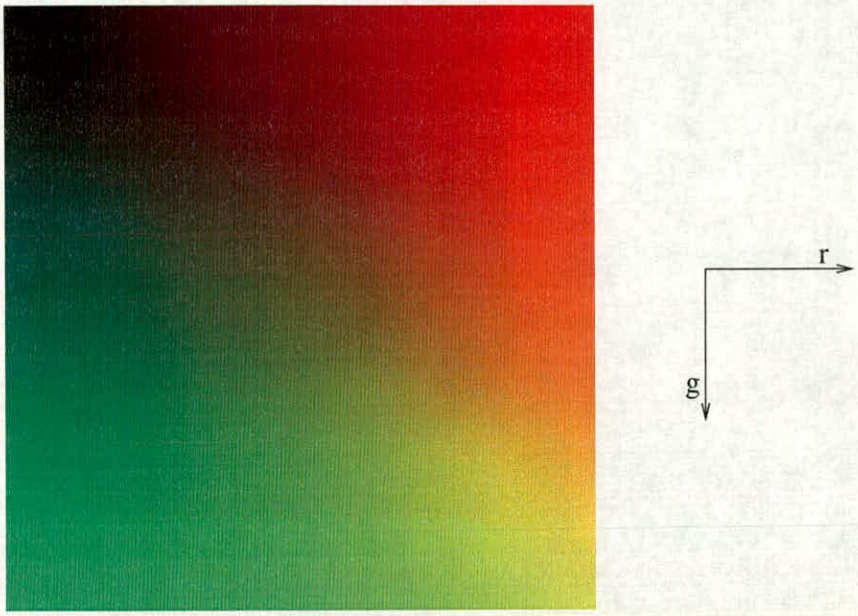


Figure 2.3: *Part of the RGB colour space. The blue component is zero for all pixels shown*

Using $x = R/(R+G+B)$ and $y = G/(R+G+B)$ means that x and y depend on the colour of the object and the colour of the light source alone: Normalised RGB space is almost immune to Lambertian (from dull surfaces) and specular reflection (from shiny surfaces) [19] as well as shadows. However, the dimensionality of the colour data is reduced, which can make segmentation more intractable. Greyscale images, for example, cannot be processed in normalised RGB space.

A slice of the RGB colour cube is sown in Fig. 2.3.

2.4.2 Hue, Saturation, Intensity (HSI) Colour Space

HSI colour space [13, 17] is a 3d non-linear transformation of RGB space. The dimensions are Hue, Saturation and Intensity, respectively.

$$
\begin{aligned}
H(R,G,B) &= \arctan\left(\frac{\sqrt{3}\,(G-B)}{(R-G)+(R-B)}\right) \\
S(R,G,B) &= 1-\frac{\min\,(R,G,B)}{R+G+B} \\
I(R,G,B) &= R+G+B
\end{aligned}
$$

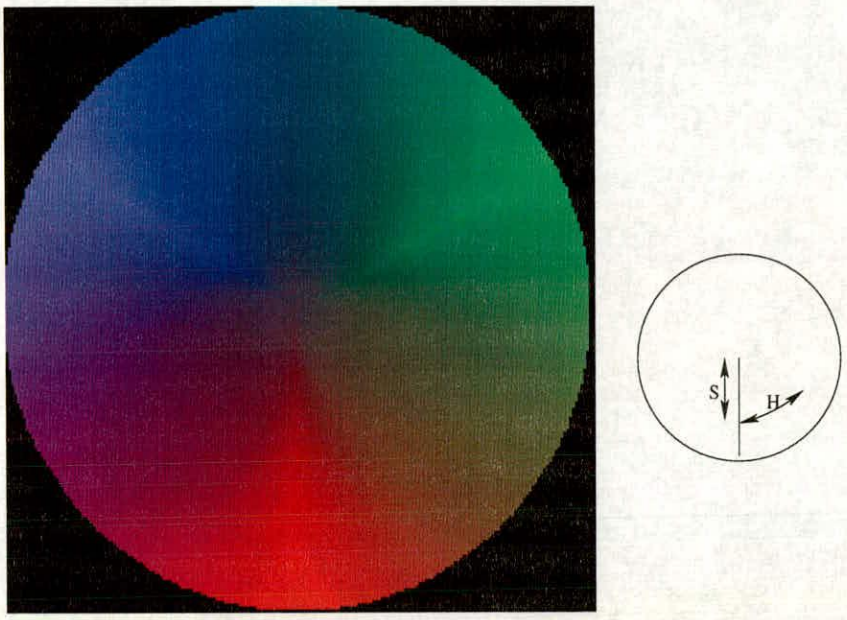


Figure 2.4: *Plot of part of the HSI colour space. The intensity is 1 for all pixels shown*

Hue is a circular dimension that represents the colour of the pixel, and corresponds approximately to the dominant wavelength of the light reflected or emitted by the object. Intensity is the average value of r g and b and saturation is the normalised minimum of the three intensities. Decreasing saturation is rather like adding white paint to coloured paint of a given hue and decreasing intensity like adding black paint.

Skarbek [13] shows that the hue dimension is not affected by illumination intensity, by specular highlights, nor by shadows. This makes HSI space immune to many of the problems of RGB space, while still providing a 3d colour space. However, the circular property of hue makes differencing slightly more complex. Also, the value of hue for a truly grey pixel is nonsensical. Techniques that use HSI colour space avoid this by processing grey areas separately from coloured ones.

HSI space is similar to HSV space (Hue – Saturation – Value) which is commonly used for image editing packages. An explanation of this space is given in the *xv* manual [18] and in Foley *et al* [19].

Since the hue dimension is circular, HSI space forms a cylinder in 3d space. A slice of this cylinder is shown in Fig. 2.4.

(a) Original (b) RGB Quantisation (c) CIE-Lab Quantisation

Figure 2.5: *Quantising an image in CIE-Lab space produces a more pleasing result than using RGB space. Source image is the Lena image. The image is reduced to five distinct colours*

2.4.3 $CIE - Lab$ Colour Space

$CIE - L^*a^*b^*$ space (the asterisks are usually omitted) was designed to be mathematically equivalent to human perception. Suppose a human decides that colour samples A and B are as different from each other as C is from D. Then in $CIE - Lab$ space, $|\overrightarrow{B-A}| \approx |\overrightarrow{D-C}|$. Like HSI space, one of the three channels represents intensity. Because this space is close to human perception, it is the best space to use when processing an image in order to improve its visual appearance, or when compressing images by discarding information that will not be detectable by the human eye.

$CIE - Lab$ space is converted from $CIExyz$ space, a linear transform of RGB space whose exact values depend on the white value of the sensor or display. For this thesis, the following conversion has been used:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0.3590332 & 0.376219 & 0.189828 \\ 0.2122671 & 0.715160 & 0.072169 \\ 0.0177579 & 0.109477 & 0.872766 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{2.3}$$

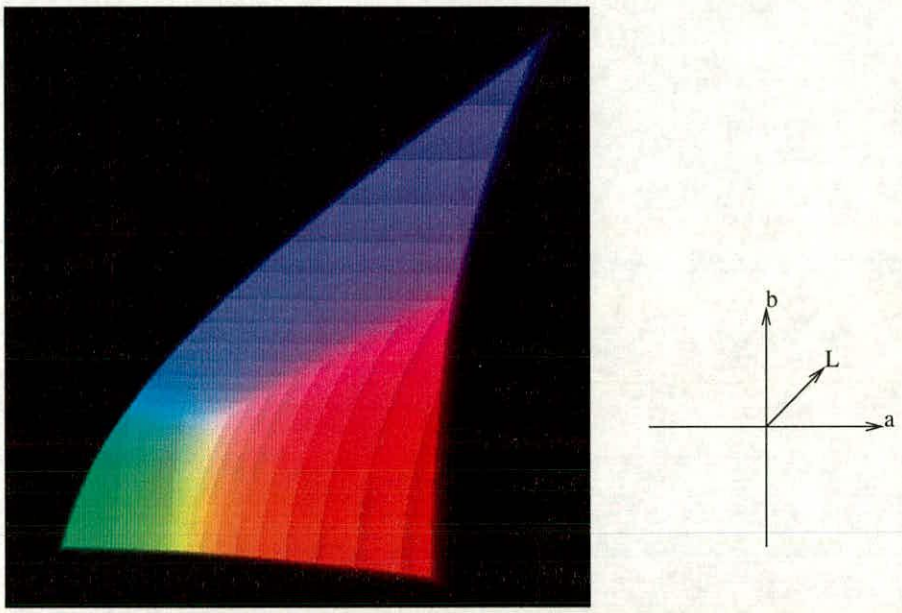


Figure 2.6: *3d plot of the $CIE - Lab$ colour space. Layers of equal values of L are shown*

$$L = 116\sqrt[3]{y} - 16 \tag{2.4}$$

$$a = 500(f(x) - f(y)) \tag{2.5}$$

$$b = 200(f(y) - f(z)) \tag{2.6}$$

where

$$f(p) = \begin{cases} \sqrt[3]{p} & \text{if } p > 0.008856 \\ \frac{7.787p+16}{16} & \text{otherwise} \end{cases}$$

for $p \in \{x, y, z\}$.

An example of the advantages of using $CIE - Lab$ space is shown in Fig. 2.5. Orchard Bouman [20] Colour Quantisation has been applied to the image to reduce it to five distinct output colours. The five output colours are chosen to best represent the distribution of colours in the image, and each pixel is set to the output colour to which it is closest in colour space. While the five output colours used in each image are not significantly different, the measure of which output colour is closest to pixels in the source image is different, so pixels are set to different output colours. Clearly, the $CIE - Lab$ quantisation (Fig. 2.5(c)) better represents the detail in the image, particularly as it preserves detail around the face.

A 3d plot of part of the $CIE - Lab$ colourspace is shown in Fig. 2.6.

2.5 Motion Picture Imaging

2.5.1 Imaging on film

Colour photography has been in existence for more than 140 years. In 1861 James Clark Maxwell projected an image of a tartan ribbon in full colour, by photographing the image on black and white film in front of red, green and blue filters, and simultaneously projecting the three images through filters onto a screen [21]. This additive technique is occasionally still used for special purpose photography [22]. Colour photography has been widely used for Motion Picture imaging for sixty years or more, since the advent of subtractive "tripack" colour film stock that can capture the three colours in a single image.

Black and White photography forms images out of light sensitive silver compounds [23, 24]. Silver Halide crystals are grown using Silver Nitrate and a combination of Bromides, Iodides and Chlorides. Light sensitive layers, with controllable sensitivities, are attached to the crystals, which are then suspended in a gelatine emulsion, and attached to a film backing [25].

When a photon of light strikes the light sensitive layer, it causes the release of a single electron, which combines with a hole in the silver halide crystal lattice, creating a separate silver ion. If enough photons (typically about twenty) strike a particular crystal and create silver ions, the crystal becomes stable in this state, and is considered to be a "latent image site."

Film developer turns all the silver halide crystals into silver metal, but it will act faster on grains that are latent image sites. The image is developed for exactly long enough for all crystals that are latent image sites to become silver, and all the unexposed to remain as silver halide. The silver halide is later removed from the film by the fixer.

The silver metal is opaque to light transmitted through the film, and thus where the original was light, the silver makes the negative appear dark.

Therefore, film sensitivity is essentially binary: If enough photons strike a crystal, or *grain*, and the developer turns the crystal to silver metal, it will become opaque, otherwise it will remain transparent. However, the larger a grain is, the more likely it is to receive enough photons to switch. Midtones are created because film has a variety of sizes of grains, so there will be areas

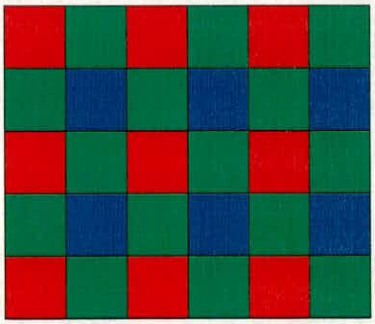


Figure 2.7: *Bayer pattern of coloured detectors for a single chip colour image sensor. 50% of the detectors are green sensitive*

where the large crystals become opaque but the smaller ones remain transparent. The area will be partially transparent and appear to be a midtone.

There are several different colour film systems [26, 14]. Typically, film is made to be colour sensitive by creating layers of crystals that are differently spectrally sensitive. These layers also contain "coupler dyes" which, during development, bind with the silver metals and become coloured. The last stage of processing is a bleach which removes the grey colour of the silver metals, leaving an image formed from the coloured dyes alone.

The larger the average size of the grain is, the faster (i.e. more light sensitive) the film will be, but the increased grain size reduces the resolving power of the film. The noise caused by the grain in film is quoted in rms granularity [27], rather than as a signal to noise ratio.

2.5.2 Digital Imaging

Unlike film sensors, a digital image sensor (CCD or CMOS) has a regular array of photodiodes (one per pixel), each of which is made sensitive to a particular colour of light by placing optical filters above them. These sensors are capable of detecting midtones, so a digital image sensor needs many fewer sensors than a film needs grains.

Colour digital cameras may use three separate sensors or a single multicolour sensor. In 3 sensor arrays, light is split into red, green and blue spectra using a prism arrangement and/or a set of filters. Each colour is then sampled on a separate chip.

In a single-chip sensor, red green and blue filters are placed over the light sensitive cells, to allow red, green and blue sensitive cells to be combined on the same chip. The separate red,

green and blue sensors are often arranged in a Bayer Pattern [28] (shown in Fig. 2.7), which has twice as many green sensors as red and blue. Post-processing interpolates the colours, filling in the missing two colours for each pixel. The regular arrangement of the photodiodes makes these sensors susceptible to aliasing effects. Because the spatial frequencies of the colour bands are different, colour band artifacts are sometimes visible in the image, caused by aliasing in the red and blue bands at a lower frequency than the green band. Merrill [29] has developed a single chip CMOS sensor that requires neither filters nor interpolation. Since the depth that a photon penetrates silicon is dependent on its wavelength [30], a single pixel can have three stacked receptors to collect each colour. Since the spatial frequency is the same for all colours, aliasing is a less severe problem.

Aliasing effects in film are not apparent due to the uneven distribution of the grains.

2.5.3 Scanning

If film is to be digitally processed, it must first be scanned to provide a digital image with which to work. It is important to scan the film at high a resolution, in order to sample accurately the shape of the larger grains, so that they can be correctly transferred into the final image.

Dust and scratches occurring on film can cause problems to image processing systems, especially if images are to be processed using noise-sensitive algorithms. Scratches can be removed using specific scratch removal algorithms [31], or more general inpainting algorithms [32] to detect the scratches and replace them with synthetically generated information.

Film grain noise can be removed by filtering. Film grain cannot be treated as signal-independent additive noise, because the amount of noise visible is strongly dependent on the average intensity of the area. Film grain noise removal is a well researched area. Chun Kit Yan and Hatzinakos [33] developed a Weiner filter to remove noise using Higher-order statistics. Their technique can also create noise, which is useful when compositing together filmed images with noise-free artificial images.

2.5.4 Motion Picture Resolutions

Table 2.1 shows the relative size of different image standards commonly used in image processing and in motion picture imaging. The number of Compact Discs required to store one

Image format	Bits per pel	Resolution (pels)	Storage requirements	
			Single frame	CDs / minute
greyscale CIF	8	288 × 352	99kb	0.25
colour VGA	24	640 × 480	900kb	2
colour SVGA	24	800 × 600	1.37MB	3
HDCAM	36	1920 × 1080	7.9MB	17
Full frame film	42	4096 × 3072	48MB	100
Vista vision frame	42	6144 × 4096	96MB	200

Table 2.1: *Comparison of different image data formats*

minute of data is detailed in the last column. The framerate in each case is 24fps — the standard for motion pictures. Typical scanning resolutions and file sizes are quoted from [1]. The most common file format for storing digital motion picture image data is Kodak's Cineon format [34]. This is a general format that can store images of any size, with any number of channels. Film is strongly log sensitive, resulting in a very high dynamic range. Therefore, film is scanned at 14 bits per channel and reduced on a log scale to 10 bits per channel. This allows all three channels for each pixel to fit into a 32 bit word.

Vista vision format (where the film is run through the camera horizontally to allow a much wider frame) is the largest format used during 35mm production. 70mm formats, including IMAX, are rarely used in standard movie-making [25].

Clearly, motion picture images are massively larger than those used for standard image processing. Algorithms that are used to process motion picture resolution images must be therefore be conservative in memory usage and must run fast enough to be practical.

2.5.5 Is Film Dying?

There is an increasingly large amount of digital imaging used in motion pictures. Current estimates are that before 2010, most motion pictures will be captured on digital cameras, digitally edited, transfered to cinemas over digital networks, and digitally projected [35]. It seems that the current limitation on entirely digital production is the quality of digital projectors. Currently, most digital movies are transferred to film in order to be projected in cinema theatres. Image capture and editing (the areas with which this thesis are concerned) are increasingly all digital. Lucasfilm's *Star Wars Episode II* [36] was filmed using the Sony HDW-F900 high resolution camcorder [37]. A comparison between shooting using this camera compared to a

	Film	**HDW-F900**
Resolution ($\times 10^6$ pixels)	12	2.2
Frame rate (max. frames / sec)	150	30
Dynamic range (bits per channel)	14	12
Max recording time (mins. at 24fps)	20	50
Typical media cost (£ ex VAT per hour)	4000	100

Table 2.2: *Comparison between digitised film images and the Sony HDW-F900 Camcorder*

standard film camera [38] (and then digitising the result) is shown in Table 2.2. Costings are provided by Panavision UK [39].

While digital cameras do not offer the same image quality as film cameras and cannot provide fast frame-rates to allow slow motion photography, they do have many advantages: Motion picture film must be kept cool, must be handled in complete darkness, and must be processed before it can be viewed. The short filming time on each reel results in wastage of film, as there must be enough footage left on the reel for each shot. Digital cameras are immune to film weave, caused by minor misregistration between frames, as well as scratches caused by the edges of the film fraying and getting caught in the camera (a film camera must be checked for such "hairs in the gate" after every take, which wastes time and film [25]). Low budget productions can re-use digital tape several times in order to save costs.

Despite the continued development of film technology, digital image quality is very likely to overtake that of film. Resolution of sensors is increasing rapidly. The Foveon f7 x3 sensor [40], which employs Merrill's scheme for sampling all three colours at a single pixel, offers an un-interpolated pixel resolution of 10.3 million pixels but achieves only 2 fps at this resolution. It is likely that these high resolution sensors will increase in frame-rates until they reach the 24fps rate of standard motion pictures. At this point, storage issues become a problem - the uncompressed storage size at this resolution is about 3.3 terabytes per hour, so large drives or tapes would be required unless a near-lossless compression algorithm could be employed.

It is probable that the use of film in motion picture production will continue for many years in special cases. Where very high resolution is required (perhaps to allow selective enlargement of part of the frame or for large format cinema such as IMAX), where high dynamic range or slow motion is required, film may prove to be the best format for many decades. Industrial latency may also play a part: Those investing millions of dollars into motion picture projects know and trust film, and may take some time to trust digital production.

While many of the images and image sequences used in this thesis are originated on film (partly due to the lack of availability of high definition digital movie sequences), the work is equally relevant to all high resolution images. Even if all movies that use digital special effects cease to use film in favour of digital image capture, most of the work presented here can be applied to digital data.

2.6 Summary

This chapter has introduced digital compositing of motion picture images. In order to composite an element in a natural image over a different background, an alpha channel and clean foreground image must be generated for the image in order to remove all traces of the unwanted background in the element. The generation of the alpha channel is in essence a segmentation problem, since the alpha channel indicates which parts of the image are required and which are not. In order to produce a convincing composite, alpha channels must include grey values to produce transparent blends of background and foreground in the final image. Segmentation algorithms that are to operate on motion picture images must perform well at the high resolution involved.

Chapter 3
Still Colour Image Segmentation

3.1 Introduction

This chapter considers previously existing techniques for still image segmentation.

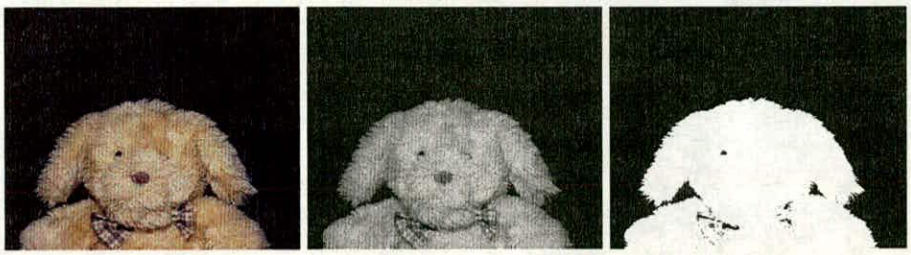
Still image segmentation techniques have been researched and developed for a number of years. Many of the algorithms are specific to particular types of images (for example Hance *et al*'s algorithm for separating tumours from images of skin [41] using colour and texture characteristics), while others are more general. Very few of these general algorithms are used in the motion picture industry. This is mainly due to the complexity of motion picture images — many of the general algorithms do not cope well with very large images containing many objects.

Several algorithms that are similar to the work presented in this thesis have been published during the period of this research. Since there is considerable overlap between this recent work and the work presented here, descriptions of these algorithms are presented together with the work described in this thesis in Chapter 6.

This chapter is structured as follows: Section 3.2 examines some of the segmentation techniques that are commonly applied to motion picture images. Section 3.3 discusses the difference between these techniques and those that follow, namely region based techniques described in Section 3.4, edge–based techniques described in Section 3.5 and texture techniques described in Section 3.6. The chapter is summarised in Section 3.7.

3.2 Motion Picture Segmentation Techniques

Many of these techniques are described in [1] and are summarised and illustrated below:



(a) Original image (b) Red channel of image (c) Thresholded image

Figure 3.1: *Example of alpha channel generation using Lumakey*

3.2.1 Lumakey

This greyscale technique is perhaps the simplest way of producing an alpha channel. The background is assumed to be darker than the foreground. Therefore, any pixel darker than a threshold value is background and therefore set to 0 in the alpha channel; any pixel brighter than a threshold value is foreground and set to 1.

The technique is illustrated in Fig. 3.1. Rather than averaging the channels together to produce a greyscale image, the red channel was used as the greyscale image, as this had the greatest distinction between background and foreground intensities.

The output shows that the border area between background and foreground has been attracted quite successfully, but there are a few areas (notably around the eye) that are erroneous. The biggest difficulty with the technique is the requirement of making the background darker than the foreground. The image in Fig. 3.1 had to be lit in order to remove all shadows cast onto the foreground, as these would be too dark to be distinctive from the background.

A more general form of segmentation by thresholding allows a weaker restriction: That no intensity level in the foreground is also present in the background. Thus, an image of a zebra against a grey background can be segmented because foreground pixels intensities are either greater than or less than any intensity present in the background, never the same.

In an automatic segmentation system, the level at which to threshold the image must be calculated automatically. Sankur and Sezgin [42] have reviewed image thresholding techniques.

(a) background image (b) foreground image (c) alpha channel

Figure 3.2: *Example of alpha generation using differencing*

3.2.2 Differencing

In this technique, a shot of the background to be removed is taken separately from the foreground shot. Any pixel that is different in the two images can then be considered foreground. An example of this technique is shown in Fig. 3.2. Holes appear in the foreground where the background and foreground images are too similar.

Although differencing allows for more natural backgrounds, it is as restrictive as Lumakey: The camera must not move between the two shots and all its settings must remain constant, as must the lighting.

Few more general algorithms rely on differencing as an appropriate technique for still image segmentation because of the amount of effort required. Advantages and disadvantages of this technique for moving data are covered in section 4.2, where more general differencing algorithms are also discussed.

3.2.3 Chromakey

Chromakey is the most common technique used to create elements with alpha channels. The object is photographed in front of a screen which is a single colour, under extremely carefully controlled illumination. This colour is unique from any appearing in the subject. The screen is typically blue, giving the technique its popular name: *bluescreen.*

Algorithms that process the image vary: The ChromaKey algorithm [43] measures the colour distance between each pixel in the image and the colour of the known background, using a

(a) original image **(b)** resultant matte

Figure 3.3: *Example of alpha generation using CFC's* Keylight *Chromakey algorithm*

hexa–octahedral colour space and distance measure. The Ultimatte algorithm (originally developed by Petro Vlahos [44] and improved by himself and Paul Vlahos [45]) assumes that the blue background screen is coloured such that there is information mainly in the blue channel and virtually none in the red or green. If the red green and blue channels for a pixel within the background area are compared, the blue channel magnitude will be greater than both the red channel and the green channel. If this is not the case, the pixel is considered to be foreground. This is equivalent to thresholding the blue channel of a pixel, where the threshold value for each pixel is the value of the green or red channel of that pixel, depending on which is the greater.

The results of using the Computer Film Company's software package *Keylight* [46] are shown in Fig. 3.3. The alpha channel is shown in more detail in Fig. 6.5. Note that, although the curtain and the flowers are not part of the actor, they will still be included in the element because they are not blue.

Photographing a scene with a bluescreen presents many problems: The backing must be lit very evenly and must be large enough to cover the whole shot (it is extremely expensive to use it for a wide angle shot, or a shot of someone moving over a long distance). No light must reflect off the screen onto the subject, otherwise that area will appear blue and will be removed. Ultimatte have developed a filter that helps to correct skin tones illuminated by the blue backing, but it is not always suitable. Ensuring that the subject has no trace of the backing colour is extremely limiting.

3.2.3.1 Truematte

The Truematte system [47] was developed by BBC Research for use in virtual studio environments: Presenters sit in small studios and virtual reality graphics add artificial elements into the scene (such as television screens and newsrooms, complete with artificial reflections).

Truematte is different from conventional bluescreen in that the backing screen is grey in colour and reflects light along the axis of incidence ('Retro-reflective'). This is illuminated using a ring of ultra bright blue LEDs placed around the camera lens. Because all of the light from the LEDs that is incident on the backing is reflected back along the axis of incidence, and therefore into the camera lens, very low intensities of light can be used. The image observed by the camera has a blue background, which can be processed using standard Chromakey techniques. This system eliminates blue light reflecting onto the subject (due to the low intensity of light used) and does not require complex lighting setups. However, it cannot (as Chromakey can) extract shadows, because the shadows cast by the LEDs are obscured to the camera lens. Again, there are geometrical restrictions on the set - if the actors are too close to the camera, they will be illuminated too strongly by the LEDs and will become transparent. If the backing screen is too far away, the intensity of light will drop off and the backing will begin to appear in the final image. The backing fabric is extremely expensive - approx. £115 per square metre [48].

3.2.3.2 Extra Channel Keying

Ben-Ezra [49] proposes a technique similar to the Truematte system. Instead of performing the segmentation by shining light onto the scene, a polarised light source is used. The foreground will tend to depolarise the light, but a sufficiently reflective background will not. The camera generates a normal image as well as a separate matte channel by splitting the image and passing one through a polarisation filter. The backing in this system is lit separately from a separate (polarised) light source, so there are fewer restrictions on camera position as there are in Truematte.

3.2.4 Depth Perception

An alternative technique for segmenting a scene is to use depth information, rather than image processing. The ZCam system developed by 3DV Systems [50, 51] measures the distance from the camera to the point in space that corresponds to each pixel in the image. The camera consists

of a standard colour CCD sensor array as well as an infra-red sensitive sensor array. A short pulse of Infra-red radiation is emitted into the scene, and the IR sensitive CCD array is made active for the same period of time. The sensor then integrates the amount of radiation received while the array is active. Objects that are close to the camera reflect back the IR radiation immediately and so a large amount of charge is accumulated on the array. Where objects are further away, the time between when the radiation reaches the sensor and charge starts accumulating and when the sensor array is turned off is shorter, so less radiation is accumulated. Thus, the resultant image from the IR CCD array is a 'range channel' where the intensities in the channel are proportional to the proximity of corresponding objects in the scene.

This depth information can also be estimated using stereo vision. The difference between images taken by two adjacent cameras will be greater for closer objects — the parallax effect. By measuring the horizontal difference in position of the same feature in two images (Chen and Liao [52] use edges), the range can be calculated.

Once the distance to each pixel has been determined, the resultant range channel has to be segmented [53]. This is a non-trivial operation, but rather simpler than segmenting a normal (intensity) image, since there is a closer correspondence between salient objects in the scene and regions in the image.

Both these techniques require that the scene be photographed with the specific intention of segmenting the sequence, a restriction that it would be useful to avoid. Stereo vision would involve the use of twice the amount of film and twice the number of cameras, which is not acceptable.

The effectiveness of depth perception algorithms in order to perform segmentation depends on the geometry of the scene: If the foreground is much closer to the camera than any part of the background, the system need not be particularly accurate at estimating depth in order to extract an acceptable matte. However, large amounts of precision would be required in order to extract a greyscale alpha channel where there are very small foreground objects (for example individual strands of hair) or where there is motion blur.

3.2.5 Rotosplining

Where no technique can separate the background and foreground, the mask must be generated by hand. In the case of a moving image sequence, there is no need to hand-draw every frame.

Typically, the edges are drawn for the first frame and then adjusted by hand to fit after a few frames. The curves for the intermediate frames are adjusted using geometric interpolation. Intermediate alpha values are created by blurring the edge of the mask. Values caused by motion blur can be simulated by averaging together the mask for two successive frames. A package such as Puffin Design's *Commotion* [10] makes this process simpler, but it remains a time consuming solution to the matte extraction problem. Intelligent Scissors is an interactive tool that can help with Rotosplining. The technique is described in section 3.5.1.

3.3 Pixel-based Segmentation

The image segmentation techniques described previously are all based on single pixel colour alone - a given pixel ij in the alpha channel is solely dependent on the colour of the pixel ij in the source image(s). Such algorithms are commonly used in the motion picture industry, since they are often efficient enough to run quickly and there is always the possibility of using a human operator to correct any errors they make

Solutions to general segmentation problems often take into account more than one pixel value at once when determining the final segmentation. Many of the techniques are variations on, and combinations of, region growing, edge detection and texture analysis, as explained below.

3.4 Region Growing

Region growing and merging algorithms combine similar adjacent areas of the image together to form larger regions. These algorithms typically have a *homogeneity criterion* that indicates whether combination is allowed to occur. A simple algorithm for region growing is outlined as Algorithm 2 adapted from [54]. Here, a *neighbour* is an adjacent pixel. Four way (North South West East) or Eight way (N,NE,E,SE,S,SW,W,NW) adjacency are typically used.

A simple Boolean homogeneity criterion $C(\boldsymbol{p}, \boldsymbol{s})$ might be $C(\boldsymbol{p}, \boldsymbol{s}) = |\boldsymbol{p} - \boldsymbol{s}| < T$ where T is some threshold, $\boldsymbol{s}$ is the colour of the seed pixel and $\boldsymbol{p}$ is the colour of the pixel under test. Ikonomakis *et al* [55] extend this metric to HSI colour space. They compare a variety of different homogeneity criteria and conclude that a modified Euclidean distance criterion produces the most accurate results:

```
Given a seed pixel s
push s onto stack
while stack is not empty
   pop next pixel p off stack
   if pixel p passes homogeneity criterion
      mark pixel as in region
      foreach neighbour r
         if r not in region
            push r onto stack
         endif
      next neighbour r
   endif
endwhile
```

Algorithm 2: *Simple region growing*

$$
\begin{aligned}
C(p,s) &= \sqrt{(d_I)^2 + d_C} < T \\
d_I &= |I_p - I_s| \\
d_C &= (S_s)^2 + (S_p)^2 - 2S_pS_i\cos\theta
\end{aligned}
$$

where I and S represent intensity and saturation respectively and θ is the difference between the hues of the two pixels (always less than 180^0) and T is the threshold

Comparing two adjacent pixels, or comparing each new pixel to the mean of the region, often produces poor results. In order for the region growing to terminate at the edges between different regions in the image, the homogeneity criterion must fail at every point along the edge, but must not be so sensitive that noisy pixels and lighting changes cause the region growing to terminate prematurely. Siebert [56] proposes a technique that correctly grows regions even where their borders are broken in places. Regions are prevented from "leaking out" through the resultant gaps. An extra homogeneity criterion is added that measures the ratio of the perimeter length to the area of the region, a property that increases suddenly when a leak occurs.

Even with such modifications to region growing, it would be almost impossible to devise a homogeneity criterion that would allow the simple region growing algorithm above to enclose the entire teddybear of Fig. 3.2(b) without including any of the unwanted background.

Rather than growing using a standard homogeneity criterion, Adams and Bischof [57] grew

regions simultaneously. Where a pixel was bordering two possible regions, it was merged with the region whose mean was closest to the pixel value. This technique has the advantage that only the required number of regions are grown (one for each seed). However, the initial seed pixels and the number of regions must be chosen carefully in order to achieve effective results.

A more robust approach is to merge regions together. Rather than attempt to enclose the entire foreground in a single region, the whole scene could be split into multiple regions that can then be combined to form a single foreground or background region. Ikonomatakis *et al* [54] grow multiple regions iteratively by finding an assigned pixel and growing a region using that pixel as a seed, repeating until there are no remaining unassigned pixels.

A simple merging step proposed by Ikonomatakis *et al* is to compare the mean pixel values in adjacent regions and merge them if they differ by less than a threshold M, where $M > T$. Regions that are smaller than a given size are likely to have been formed by noise and can be merged with the neighbouring region with the closest mean.

Split–and–merge algorithms work in a similar way. The image is considered to be one region, and if it fails a homogeneity criterion it is split into four subregions. Splitting continues recursively until all regions pass the homogeneity criterion. In the subsequent merging step, neighbouring regions that pass the homogeneity criterion when united are combined.

Yang and Lee [58] have a more efficient split–and–merge algorithm. Conventionally, regions are only subdivided into equal sub-regions. Yang and Lee attempt to subdivide by thresholding the block. If the threshold–based segmentation does not yield separate coherent regions, the normal subdivision technique is used instead.

All these region–based techniques provide crisp segmentations. Where there are poorly defined edges, fuzzy techniques can give better results. Some fuzzy algorithms (such as that by Qing *et al* [59]) use intensity levels alone, rather than combining their results with spatial information. Thus, they are similar to the Chromakey algorithms presented in Section 3.2.3. Karmakar and Dooley [60] incorporate a spatial membership function in order to segment into spatially distinct regions.

3.5 Snakes and Edges

Snakes, the popular name for *Active* or *Dynamic Contours*, are models that can find objects in images based on their edges. An edge in an image corresponds to a sudden change in intensity or colour. Thus, they can be found by differentiating the image. However, the border along an object may have many false edges, and there may be parts along the object border where the edge is missing. The active contour will pick up a consistent edge.

The first step is to position (by hand or by an alternative technique) a line that is quite close to the expected edge. This line is the initial position for the contour. Two types of forces then act on the contour. The *internal forces* act to reduce the curvature on the contour. That is, they try force it to become a circle. The *external force field* attempts to push the curve towards local minima. Typically, the external force field is created by inverting the differentiated image, so that edges form the local minima. The external energy field need not be an edge – Ray *et al* [61] use an external energy field derived from AM and FM frequency components in the image.

The position of the contour is updated by allowing these two opposing forces to act on the contour, causing it to accelerate and therefore move in certain directions, until it settles in equilibrium, where there is no significant force on the snake. The snake will generally lie along edges, and will enclose the area to be segmented.

Fig. 3.4 shows a snake fitted to the boundary of the *Gema* image. This snake is based on the algorithm of Lobregt and Viergaver [62]. Here, a snake is represented as a series of straight lines connecting points. The forces act only on the points. The internal force on a point is derived from the angle between the two lines meeting at the point, while the external force pulls the point towards the nearest area of high image gradient. In Fig. 3.4, the image has been converted to greyscale for processing (there are, however, adaptations to snakes for colour and other vector-formed images [63, 64]).

In Fig. 3.4, the snake has settled along the approximate boundary of the outline. The snake has not been able to follow the individual strands of hair on the top of the head and has been confused by the edge caused by the white band on the t-shirt. There are also many places in the image where there is no real "edge" as such (for example the area of semi-translucent hair on the left hand side) where a snake cannot accurately describe the position.

A further difficulty with using snakes on high resolution images is the initial need to place the

Figure 3.4: *Linear snake fitted to the boundary of the* Gema *image*

snake close to the edge. In the case of the *Gema* image, the snake must be placed within 20-30 pixels of its final destination. Placing the snake further away reduces the chance of it being attracted to the correct edge contour because it is so far from the minima in the external energy field that it is not attracted towards it. This restriction means that the initial snake would have to be placed with so much accuracy that there is little gain by allowing the snake to move. However, applying a snake to a low resolution image first, and then gradually increasing the resolution in order to reposition it, may relax the requirement for accurate positioning of the snake.

Active Balloons may help to solve the accurate positioning problem. These are a slight modification of Active Contours. Again, there is an internal force that forces the contour to be smooth, and an external force that attracts it towards edges in the image. However, a third *inflation* force is added. Xiaobo Li and Kiankang Wang [65] dynamically calculate the inflation force to prevent the contour from bursting out of areas where the limiting contour is poorly defined. Using this system, the balloon can be initialised some distance inside the final position, and it will expand until the edge is reached.

While balloons may be a more suitable technique for applying active contours to high resolution images, adapting either type of active contour to be sensitive to hairs and translucent areas is a much more complex problem.

3.5.1 Intelligent Scissors

Intelligent Scissors is an interactive technique for placing a contour in an image. Instead of the user first generating a contour and then allowing it to settle along an edge, and correct it if it failed to find the correct edge, the Intelligent Scissors contour realigns itself (or "snaps" to the edge) as it is being drawn. The algorithm was presented by Mortensen and Barrett in 1995 [66] and refined by them in 1999 [67]. In their later work, the image is first segmented into many small regions using the "tobogganing" approach. This is generated by following the line of steepest descent from a pixel until a local minima is reached. All pixels that descend to the same local minimum are allocated to the same region. These regions are then converted into a region adjacency graph [68] where the nodes on the graph represent each region in the image, and edges exist between two nodes if the corresponding regions are adjacent. Each edge is assigned a weight according to the gradient across the edge.

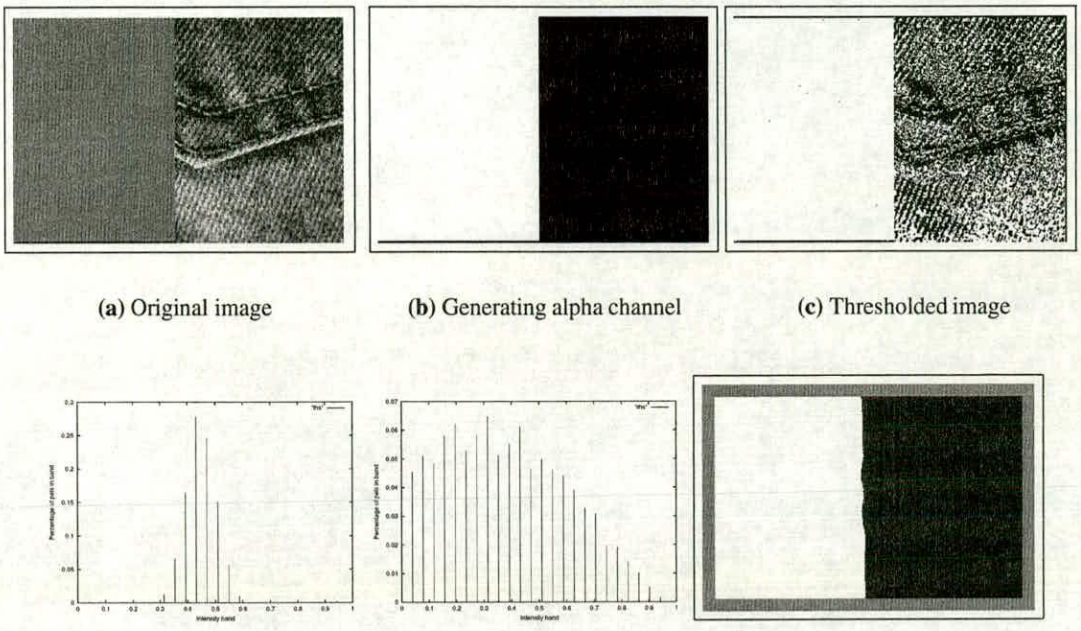
To use Intelligent Scissors, an "anchor point" is first defined by the user, usually by clicking with a mouse on the image. As the mouse pointer moves, the least cost path across the region adjacency graph to the anchor point (that is, the path that has the smoothest gradient) is calculated and displayed. The user can then click again to set the path, and to begin defining the next section.

This reduces the process of defining an edge in an image to a small number of mouse clicks. A further extension to this algorithm was presented by Mortensen in 2001 [69] that in some cases can define an edge with just one click - where there is one clear edge present, the path will follow it automatically. The user need only intervene where edges become undefined or where there are junctions.

3.6 Textures

Texture segmentations algorithms segment images based on the statistics of local areas. Texture analysis techniques can be divided into two broad categories [70]: Those that take account of the spatial arrangement of pixels, and those that consider only the statistical values.

Consider the image of Fig. 3.5(a), showing an image generated using image 137015 from the Corel Photo Gallery [71]. The left hand side is Gaussian noise, with a similar mean to the denim. The best segmentation achievable by thresholding is shown in Fig. 3.5(c). Because



(a) Original image **(b)** Generating alpha channel **(c)** Thresholded image

(d) Histogram of left side **(e)** Histogram of right side **(f)** Resultant segmentation

Figure 3.5: *Histogram based texture segmentation*

some intensity values are present in both regions of the image, there is no simple threshold technique that will correctly segment the image.

Using texture analysis, segmentation is possible. A statistics-only technique would be to use histograms. Histograms of the left and right side of the image are shown in Figs. 3.5(d) and 3.5(e). 25 bins have been used to form these "reference" histograms. Segmentation can proceed by taking each small square section of the image in turn, generating a histogram for it, then treating the 25 values produced as a vector in $\mathbb{R}^{25}$. Taking the Euclidean distance of this region to each reference histogram gives measures of similarity of this region to each reference texture. The pixel at the centre of the square section is assigned to the most similar region. The result of this algorithm is shown in Fig. 3.5(f). This segmentation is significantly better. Note that the edge, however, is slightly ragged. This is common in texture analysis techniques because, by its nature, texture must be taken over a small area of the image. If that area includes more than one region, spurious statistics result. The grey border is due to the size of the search region that prevents results from being taken too close to the edge of the image.

Note that texture analysis does not provide a means for segmentation directly; rather it produces

a new space in which to perform existing techniques. Fig. 3.5(f) was produced effectively by a texture-space thresholding. Techniques such as region growing and edge detection can be applied to texture-space to perform segmentation.

Ohta *et al* [72] describe a technique that performs region splitting using histograms. The algorithm starts with the whole image being assigned to the same region. Regions are split if the histogram produced from the region has multiple distinct maxima, then an intensity level between each maximum is used as a threshold to partition the region into two sub-regions. Threshold levels that do not produce spatially distinct regions are rejected.

Fig. 3.5 uses greyscale histograms. It is possible to use colour histograms. To form a colour histogram, the image is quantised to a small number of colours and the percentage of pixels in each possible quantised colour is calculated. This can produce very large data sets. Producing a histogram to the same resolution as that in Fig. 3.5 requires $25^3 = 15625$ bins. This is a massive vector and it would be extremely computationally intensive to operate with. Ohta *et al* instead use the Karhunen Loeve transform to reduce the colour space from $\mathbb{R}^3$ to one in $\mathbb{R}^2$. The third ordinate contains insignificant information and can be discarded. Tan and Kittler [73] use a similar approach, approximating the $\mathbb{R}^3$ histogram as three $\mathbb{R}^1$ histograms taken along the principal axes.

Rather than using Histograms, Deng and Manjunath [74] use a value related to the local variance called the J value. J is used as a proximity measure for simultaneous region growing in a method similar to Adams and Bischof [57].

Other techniques use higher order statistics in order to provide better segmentation results. Gu *et al* [75] compared different higher order texture analysis techniques to perform cloud recognition. Non-spatial techniques are appropriate in this case, since recognition must be scale and rotation invariant. Gool *et al* [70] summarise various work that suggests third and higher order statistics are not helpful in texture discrimination, since the human eye cannot distinguish between two textures that differ only in their second order statistics. This result suggests that general segmentation algorithms will not be significantly improved by using higher order statistics. Only in specific cases where information about the background and foreground are known (such as in cloud recognition) are such statistics likely to improve results.

The texture analysis schemes presented so far do not take account of the *shape* of the regions, only the local statistics. This may be an advantage or a disadvantage: The invariance of Histo-

gram analysis to orientation of the texture will allow the same pair of jeans to be segmented as a single object, even though there may be folds and creases in the denim. However, they may be a case equivalent to the left side of Fig. 3.5(a) being a mirror image of the right. Then, the histogram on either side would be identical and histogram analysis would fail even though they would clearly appear as different regions.

The DCT (Discrete Cosine Transform) is a variant of the Discrete Fourier Transform that has many applications in image processing, most notably in image compression [76]. The DCT represents an image (or often small blocks of an image) as a set of basis functions that record the strength of spatial frequencies in the image in given directions. For a $N \times N$ block the $u \times v$ DCT coefficients are [77]:

$$\mathrm{DCT}(u,v) = \gamma(u)\gamma(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} I(x,y) \cos\left(\frac{(2x+1)u\pi}{2N}\right) \cos\left(\frac{(2y+1)u\pi}{2N}\right) \tag{3.1}$$

$$\gamma(u) = \begin{cases} \sqrt{\frac{1}{N}} & \text{if } u = 0 \\ \sqrt{\frac{2}{N}} & \text{otherwise} \end{cases} \tag{3.2}$$

Thus, the DCT co-efficients for a given area of the image code statistical as well as spatial information. Wei [78] segments an image by calculating for each pixel p the DCT coefficients of a 7×7 block of pixels surrounding the pixel p. Since the amount of information produced is large (51 coefficients including the position of the pixel), PCA is used to reduce the dimensionality. K-means [79] is then used to segment the image into two regions. Since the pixel co-ordinates are included in the space along with the DCT coefficients, spatial separation of the two regions tends to be maintained.

Gabor filters are an alternative frequency sensitive analysis tool. Like the DCT, Gabor filters are similar to the Fourier Transform, but act only over a small Gaussian window. Thus, they can give local information. They are used by Mital [80] for texture segmentation. A typical 2d Gabor filter is given by

$$h(x,y) = e^{-\frac{1}{2}\left(\frac{x^2}{\sigma_x^2} + \frac{y^2}{\sigma_y^2}\right)} \cos(2\pi u_o x)$$

where σ_x and σ_y are the standard deviations of the Gaussian window and u_o is the frequency of the filter. Mital applies 16 filters to each area of the image, using varying angles of rotation and varying frequencies. Thus, each location in the image has 16 coefficients, which can be grouped to segment the image.

A similar approach is taken by Acharyya and Kundu [81]. They use wavelet theory to form multiple images at different resolutions, using progressive low pass filtering. For each pixel, the intensity value for the pixel in each of the filtered images is formed into a feature vector. These vectors are then classified using a K-means classifier.

These systems derive texture features using low-level operations. Co-occurrence matrices measure the relative spatial frequencies in an image. A co-occurrence matrix for an image with I intensity levels is of size $I \times I$. Every pair of pixels p_1 and p_2, separated by a fixed distance d and angle Θ, are scanned in turn. If i is the intensity value of p_1 and j the intensity of p_2, then entry ij in the co-occurrence matrix is incremented. Therefore, a finished co-occurrence matrix counts how often pairs of pixels of given intensity values, separated by d occur in an image. Lam [82] uses a different formulation of Co-occurrence matrices to derive gradient information from the image.

Since an $I \times I$ matrix is often too large for analysis, second order statistics are extracted from the matrix. Gotlieb and Kreyzsig [83] analyse six different measures and their effectiveness for classifying textures.

Aksoy and Haralick [84] compare measures derived from co-occurrence matrices with the Line–angle–ratio measure. Here, lines are found in the image using edge detection. Where two lines intersect or nearly intersect, the ratio is taken of the mean intensity outside of the lines to the mean intensity of the area bounded by them.

3.7 Summary

This chapter has presented a variety of techniques for the segmentation of images. Standard segmentation techniques can generally be grouped into those that consider regions of pixels, those that search for edges in images, and those that use statistical operations to sense texture. Many of these standard techniques cannot be readily applied to the task of alpha channel estimation of motion picture images. Texture techniques do not segment particularly well at

edges, since texture must be measured over a small area of an image, which means that the texture must be sampled at both sides of the unknown edge. Region growing and contour based techniques assume that the edge is very sharp, which will not be the case for a blurred image.

Techniques currently employed on motion picture images require more specific environments (chromakey and lumakey require a certain background and difference matting requires the background to be photographed separately) but have the advantage of simplicity, speed and the ability to produce accurate information even where edges are blurred.

Chapter 4
Moving Image Segmentation

4.1 Introduction

This chapter considers previously existing techniques for moving image segmentation.

Since a moving image is a sequence of discrete still images, it is possible to segment moving images by applying a still image technique. However, much information can be derived from analysing more than one frame at once, and using information about the differences between them in the segmentation. This chapter examines existing techniques for segmenting image sequences.

This chapter is structured as follows: Section 4.2 considers techniques that can segment each frame of a sequence by comparing it to an image of the background. Sections 4.3 and 4.4 discuss the use of Block Matching and Optical Flow motion estimation algorithms, respectively, to track the foreground through a sequence, given an estimate of the foreground in an original frame. Section 4.5 examines techniques that segment using estimates of motion directly, without using an initial segmentation. Section 4.6 examines the extension of the active contour techniques discussed in section 3.5 to moving images. Similarly, section 4.7 examines the extension of region growing to moving images. The chapter is summarised in section 4.8.

4.2 Difference Matting

Difference matting is a time-domain version of the still image differencing technique described in section 3.2.2. Again, frames are compared to one or more reference images of the background. However, in this case it may not be necessary to photograph the background independently, especially if the camera and background scene are static. Special Effects compositors often build reference images manually by assembling together different parts of the background image from different frames of the sequence. However, it is possible to generate reference images automatically by processing the sequence. Median averaging the sequence will produce a

single correct background image if the foreground covers no part of the image for more than half of the sequence [85]. Where the sequence is very long this becomes computationally expensive, so a low pass filter is more appropriate [86]. This generates a different reference image for each frame of the sequence and so it can work where the background is slowly evolving. Boninsegna and Bozzoli [87] use a Kalman filter in place of a low pass filter, which makes it robust to sudden changes in the background, but not to significant camera motion.

Where the foreground object casts a shadow on the unwanted background, this will appear as a difference between the reference image and the current image, and so may appear in the final foreground object after differencing. If the shadow is treated as part of the element, it will cause parts of the unwanted background to appear in the final composite. Horprasert *et al* [88] detect shadows as a special case by assuming that they are an area where the current image is the same as the reference background image, but where each channel is multiplied by a constant gain n, where $n < 1$. The technique of Horprasert *et al* also automatically selects the level at which the difference image should be thresholded.

An alternative to creating an average background frame and then comparing each frame to this background is to compare the frames to each other successively [89]. Rather than comparing frame $n+1$ to the reference image, frame $n+1$ compared to frame n. If the background is static and the foreground is always moving, then any significant differences between the two frames will be due to this moving foreground. However, some pixels will be different due to noise or flicker. Lee and Kim [90] apply an adaptive statistical function to improve performance with noisy images, which would certainly be useful for grainy motion picture sequences. Another problem, described by Meier and Ngan [91], is that the foreground must move in its entirety - if someone is standing still but moving their arm, then only the pixels around their arm will be different in the two frames, so the rest of their body is liable to disappear. This can be avoided by carrying forward the previous segmentation result. If a pixel does not change between two successive frames, it is treated as being the same classification as before, rather than always being classified as background.

The results of difference matting are always likely to be noisy, with parts of the image showing movement where there is none and *vice versa*. Vass *et al* [92] use a split-and merge region growing technique as well as differencing to perform their segmentation. Now, any region for which *any* pixel within the region is different is considered to be moving.

4.2.1 Differencing with Moving Backgrounds

The above techniques assume that the background is static. The camera must not move, pan or zoom, and the background must have no moving objects. Where the background is uniform but the camera moves, it may be possible to build up an oversized picture of the background (a "background mosaic") and then warp this image using planar perspective transforms for each frame so it corresponds as much as possible to the image. By differencing each frame to this warped background, the foreground area can be recovered. Szeliski [93] presents methods for background mosaic construction.

In order to use this technique for segmentation, the foreground must be removed from each frame before it is combined. The pixels in the background that are occluded by the foreground, and therefore missing from each frame, will be filled in from other frames as the foreground and background move over each other.

Hidalgo and Salembier [94] use this system to segment images. Once the background mosaic has been generated, it is warped to fit the current image and differenced with the current frame. To obtain a solid mask, the watershed region growing algorithm is applied.

4.3 Block Matching

The requirement of having a still (or at least homogeneously moving) background region can be avoided using block matching as a basis for moving image segmentation. Suppose that the segmentation for frame n is known (perhaps generated by a still image segmentation technique). If the foreground does not change significantly between frames, it can be located in frame $n+1$ simply by finding the best match for the blocks in the foreground in frame n to frame $n+1$.

Lee *et al* [95] use this method to perform segmentation for video coding. For each pixel that is part of the foreground in frame n the best match for a user-specified block in frame $n+1$ is specified.

Lee *et al* use the Summed Absolute Difference (SAD) as a similarity measure. To compare a block A extracted from frame n to a candidate block B extracted from frame $n+1$, the SAD is given by

$$SAD = \sum_{ij} \left(\left| A_{ij}^r - B_{ij}^r \right| + \left| A_{ij}^g - B_{ij}^g \right| + \left| A_{ij}^b - B_{ij}^b \right| \right) \cdot (\alpha_{ij}! = 0), 0 \leq i \leq m, 0 \leq j \leq n$$

where m, n is the size of the block, and A_{ij}^r indicates the red channel of pixel ij in block A. The use of the alpha channel α in the above equation ensures that the only pixels in frame n that are included are those which originated from the foreground. Blocks close to the edge of the foreground area would otherwise contain part of the background, which would result in a false match.

To classify a single pixel p in frame n, the similarity measure is calculated between each block B in frame $n + 1$ and the block of pixels A around p. The block that gives the best possible match to A is taken to be the position of pixel p in frame $n + 1$. This pixel is then marked as foreground in the alpha channel for frame $n + 1$. The process is repeated for all foreground pixels in frame n.

Where the foreground is subject to significant lighting changes or non-translational movement such as rotation or deformation, block matching may fail because the original block does not exist in the new frame. One solution to this is to transform the image into a different domain. Antoszczyszyn [96] performed block matching on PCA transformed images to avoid poor results from noise or from lighting changes.

Block matching is extremely computationally expensive. Lee *et al* propose a technique to reduce slightly the number of calculations required. Section 7.5.2 considers computational complexity of Block Matching in more detail.

The *Autokey* system developed by Mitsunaga *et al* [97] also uses block matching to perform segmentation on image sequences, but is more resistant to false matches and has a lower computational cost. Rather than matching every pixel in the foreground object, only those objects on the perimeter that correspond to points of high curvature (corners) in the alpha channel are used for block matching. As with Lee *et al*, the alpha channel is used during block matching to prevent matched blocks containing unwanted background pixels. The matched points in the next frame are then connected to form a continuous boundary and the space the boundary encloses is marked as foreground.

The Autokey system is capable of estimating alpha values. It assumes that the edges are very sharp with respect to the background (that is, the background and foreground are relatively uniform in the region of the edge). Thus, any change in pixel intensities will be due to a transition between background and foreground. The first step is to edge detect the image in a projection of colour space chosen to best represent the transition between local foreground and background colours. The boundary found by the block matching step is assumed to have an alpha value of 0.5, and a contour C_M initialised to this line. Two further contour lines C_B and C_F are placed alongside C_M, at a varying separation distance according to the image gradient. Points along C_B and C_F are given an alpha values of 0 and 1 respectively. The pixels between C_F and C_B are assigned alpha values using interpolation.

Autokey does not (as bluescreen systems do) generate alpha values according to actual pixel colours. By assuming that an edge is a uniform, monotonic transition between background and foreground, complex boundary regions, such as those with fine hairs, will not be segmented correctly.

4.4 Optical Flow

Optical flow [98] is based around the assumption that the intensity of an object will not change as it moves between frames in an image. That is,

$$\frac{dI}{dt} = 0 \tag{4.1}$$

where I is the intensity of the object. The chain rule can be applied to equation 4.1 to model pixel motion

$$\frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t} = E_M \tag{4.2}$$

Where E_M is ideally zero. This partial differential equation is solved for $\frac{dx}{dt}$ and $\frac{dy}{dt}$, which give the horizontal and vertical components of velocity respectively. There will be many solutions of

$\frac{dx}{dt}$ and $\frac{dy}{dt}$ that will not represent the true motion of the object. Horn and Schunck [99] solve this equation by assuming that the motion varies gradually over the image. A smoothness constraint that prevents neighbouring pixels having significantly different motion vectors is employed:

$$E_S = \left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 \tag{4.3}$$

where $u = \frac{dx}{dt}$ and $y = \frac{dy}{dt}$. The motion field can then be calculated by minimising the combined error function $E = E_M^2 + \lambda E_S$.

Optical flow can be used either in place of Block matching to identify parts of the current frame as background or foreground, or it can be used as the basis of the motion estimation techniques presented in the next section.

The suitability of optical flow to motion picture image sequences is considered in section 7.5

4.5 Motion Estimation

The previous techniques perform segmentation by comparing individual pixel values and intensities between two frames, either by using simple change detection, or by locating the same area of image in the new frame. Where there is interframe movement, the movement itself can be used to form the segmentation.

Moscheni *et al* propose a solution to segmentation using this approach [100]. If the background is approximately planar relative to the position of the camera, there will be two different motions in the image, one corresponding to the background, or *global* motion, and the other corresponding to the foreground, or *local* motion. The global motion will generally cover a larger area of the image than the local. Two distinct motions can be derived using a clustering technique such as K-means, and those pixels that correspond to the smaller of the two marked as foreground.

Moscheni *et al* use block matching to calculate the motion field, but their technique could equally be applied to optical flow techniques. False matches and outliers are a problem in either case. Haung *et al* [101] use optical flow to generate a motion field, from which outliers are removed by using a Robust Genetic Algorithm. Galić and Lončarić [102] combine the

results of optical flow with image intensity, and segment the combined space using K-means. The addition of the intensity information helps overcome problems from spurious values in the motion data. Erhan Eren *et al* [103] use spatial constraints to avoid problems with outliers by dividing each frame into multiple regions (using region growing techniques) and then merging regions together that have similar motion characteristics.

The Optical flow equation (Equation 4.1) assumes only one motion at each pixel. Where there is transparency (so that the alpha value for a pixel will be between 0 and 1) there will be two motions occurring at each pixel location — one representing the background motion and one representing the foreground. Toro *et al* [104] adapt the optical flow equation to permit two locally constant motions at each pixel location. Given the two resultant motion fields, Expectation Maximisation is used to generate motion models for the scene. A set of validity maps mark which pixels belong to each motion model. Because there can be two motions for a single pixel location, a pixel can belong to more than one motion model. Expectation Maximisation iterates between calculation of the parameters of each motion model, and calculation of the validity map for each motion model. These validity maps can be used as a final segmentation of the image.

4.6 Active Contours

Active contours (Snakes) can be extended to segment sequences as well as still frames. Recall that an active contour is initialised close to the edges of an object and adjusts itself to lie along the edge using energy minimisation. If the edge of the object moves only slightly, then the final position of the contour in frame n can be used as the initial position for frame $n+1$. With large inter-frame movement, this technique fails. Bascle *et al* [105] incorporate motion estimation into their system. A 2d affine motion model is derived from the motion field and used to update the position of the snake between frames.

Peterfreund [106] takes a different approach. A normal snake settles when the difference between the internal forces and the external energy field derived from the gradient of the image is at a minimum. Peterfreund adds a term to this minimisation which is the difference between the motion field in the image and the interframe motion of the snake. This allows tracking of objects where the edges are not well defined.

Rather than fit an active contour to each frame separately, Cohen and Cohen [107] fit an active

surface to the whole sequence at once. They simultaneously evolve a 2d snake on each frame, allowing the snakes from each frame to interact with each other. Their technique is particularly suitable for volumetric data, but could equally apply to a 2d image sequence. Hall and Jones [108] have applied active surfaces to film sequences. Contours are specified for keyframes in the image sequence (*i.e.* a contour is drawn every few frames), which are used to initialise the surface. The contours on intermediate frames are initialised using geometric linear interpolation between the nearest keyframes. The surface then evolves to fit the edges in the sequence.

Where the edges are very poorly defined, but approximate shape is known, a model based approach may produce good results. Baumberg and Hogg [109] and Peacock [110] use Eigenshape models to track pedestrian motion. Models are represented as sets of parameterised curves. A training set of possible outlines is generated, which is used to generate an eigenspace of outlines. These models are fitted to each frame of the sequence. Wachter and Nagel [111] also track humans, but use a 3d model of a human being and, given knowledge about the degrees of freedom of a human, attempt to fit their model to the edges in each frame of the sequence. Along with an alpha channel, Wachter and Nagel's system produces an estimation of the pose of the person, which could be used for animating computer generated actors that must interact with real actors.

Model based systems suffer from a lack of precision when segmenting, as a model that would be accurate enough to extract detail such as the folds of cloth or strands of hair would have too many degrees of freedom to calculate efficiently. It is likely that a different model would be required for each type of subject.

4.7 Motion-based Region Tracking

In most cases, a region of homogeneous intensity and colour in an image corresponds to a part of a single object in the scene. This is the basis behind segmentation using region growing, presented in the previous chapter. This technique has been adapted to moving sequences.

Deng and Manjunath [74] simply use block matching to locate the seed points of the regions in the next frame, and then grow regions from there.

Gu and Lee [112] segment each frame into regions and then use motion estimation to build a

correspondence between regions in the current frame and regions in the reference frame, and then assign the regions in the current frame to background or foreground, according to the classification of the corresponding region in the reference frame.

Dongxian Xu *et al* [113] combine region growing with tracking in the sequence. A contour is drawn approximately on the first frame, which is then segmented into regions. Any regions totally enclosed by the snake are considered to be part of the segmented object, *i.e.* the foreground. Any region that is intersected by the contour is included if a probability criterion is met. The contour is then updated to be the perimeter of all regions that are included in the foreground object. For subsequent frames, the interframe difference is edge detected, and this image XOR-ed with the contour from the previous image to form a contour for the new frame. This is then updated by using regions. Since the edges of the regions tend to be inaccurate, an active contour model is used to smooth the contour at each stage.

4.8 Summary

This chapter has presented a variety of techniques for segmentation of moving images. Many techniques work by generating an estimate of the background and using this to identify the foreground areas of the image. Other techniques (such as the active contour techniques) allow human–assisted single frame techniques to be applied to image sequences with little or no further human interaction. A third group of techniques, the motion segmentation techniques, achieve segmentation based on the differences between frames, and cannot be applied to single frame segmentation.

As for the single frame segmentation techniques of the previous chapter, very few algorithms are able to process moving image sequences and produce alpha channels. As before, contour and region based techniques do not readily adapt to soft, blurred edges. Block matching and smooth optical flow suffer the same lack of precision seen in the texture–based techniques of the previous chapter.

Chapter 5
Alpha Channel Estimation for Single Frames

5.1 Introduction

This section presents a group of algorithms that produce alpha channels and clean foregrounds from still images with varied backgrounds. The chapter describes progressively more complex algorithms. The simpler algorithms are presented in order to demonstrate (by their failings) the motivation behind the more advanced techniques.

Several algorithms that are similar to those developed in this thesis have been reported by Ruzon and Tomasi [114], by Chuang *et al* [115] and by Berman *et al* [116]. All of these algorithms were published while the research for this thesis was underway. The work presented in this chapter has not been influenced by these algorithms. However, an analysis of these algorithms is presented in the next chapter, as are novel adaptations to them.

This chapter is structured as follows: Section 5.2 describes the hint initialisation required for these algorithms. By rearranging the compositing equation, section 5.3 shows how alpha channel estimation can be approached as the problem of estimating clean background and foreground colours. Sections 5.5 to 5.9 introduce progressively more advanced clean colour estimation algorithms. Sections 5.10 to 5.12 describe the Principal Axis algorithm that represents a major contribution of this thesis. Section 5.13 shows how lighting from behind the foreground (*backlighting*) can cause algorithms to fail and proposes a novel solution to this problem. The chapter is summarised in section 5.14.

5.2 The Need for Human Interaction

All of the techniques described here and in the next chapter need a similar human-created input. Most motion picture images are extremely complex, in that they will contain a large number of regions with very different colours and textures. There will also be a large number of edges.

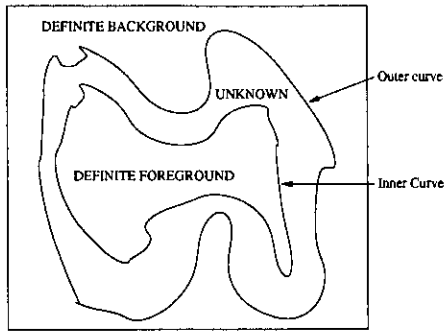


Figure 5.1: *Curves used for input to Corel* KnockOut

These regions and edges could be part of the background or the foreground, or they could be part of the transition between the two. Deducing which parts of a still image are to be retained (*i.e.* are part of the foreground) and which are to be discarded is extremely complex. Given an image containing multiple people, no simple algorithm could deduce which of the people are to be retained and which to be extracted, without some form of additional input.

A suitable form of human input is to provide a rough, hand segmentation of the image, into three types of area: Definitely background, definitely foreground, or unknown. The unknown area usually forms a band between the known background and foreground areas, and is the only one which requires processing. Pixels within this area may be part of the background, the foreground, or maybe a combination of the two, and must be assigned an alpha value accordingly. This type of input has two advantages: Firstly, classification can be achieved by comparing parts of the unknown area to the definite foreground and background. Secondly, those parts of the image that are marked as definite do not need to be processed, saving a massive amount of processing. (Only 7% of the 2.7 million pixels in the *Gema* image are marked as unknown for the results presented in this thesis.)

There are various different techniques for indicating which pixels are definite and which unknown. Corel's *KnockOut* [116] uses two curves. Everything closer to the edge of the image than the "outer curve" is definite background, that within the "inner curve" is definite foreground, and the area between the two curves is the unknown area, as shown in Fig. 5.1.

Adobe's *Photoshop* requires the user to draw over the image using standard painting tools to indicate the unknown and foreground areas (the definite background is the unmarked area). Evening [117] gives advice for the construction of alpha channels in *Photoshop*. Ruzon and Tomasi's

approach [114] requires the user to modify a copy of the image, colouring the background and foreground distinctive colours. These areas can be detected by comparing the modified image to the original.

Gu and Lee [118] perform standard binary segmentation using a similar foreground / background / unknown scheme. Pixels in the unknown area are classified as background or foreground using morphological segmentation. To create their unknown area, they require a user to mark a single edge line that initially forms two complementary regions, foreground and background. These are then eroded away to form an unknown area. The user controls the amount of erosion ensuring that the unknown region entirely contains the edges of the object.

All of these techniques for user input are in effect equivalent, and the relative merits of these are not important for this thesis.

Tan and Ahuja [119, 120] have developed a system to make the generation of hint images easier. Their technique segments the image into multiple regions, and then uses freehand sketches (often a few clicks and lines on the image) to combine the regions together. In the vicinity of the sketch, finer segmentation is undertaken in order to build up an accurate hint image.

5.2.1 Creating a *Hint Image*

A simple way of producing the required input without specialised software is to use a multi-layer image editing package, such as *Gimp* [121] or *Photoshop* [11]. The original image is loaded into the package, a new layer created, made to be semi-transparent, and then drawing tools used to draw over the image on the new layer. Requiring the background area to be marked over in black, the foreground white, and the unknown area some shade of grey is most suitable, as it adapts well to the moving image segmentation techniques presented later in this thesis. Once this layer has been created, it is saved. Note the original image is left untouched, as the editing takes place on a different layer within the package. Fig. 5.2. Shows an example "hint image" created in this way.

5.3 Alpha Estimation by Inverting the Compositing Equation

Classifying an image in order to produce an alpha channel is slightly different from standard classification [79], in that it does not involve a crisp classification, assigning each pixel as either

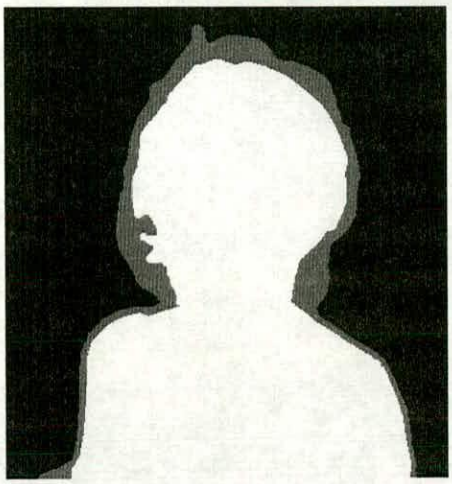


Figure 5.2: *An example "hint image" used to indicate background, foreground and unknown areas*

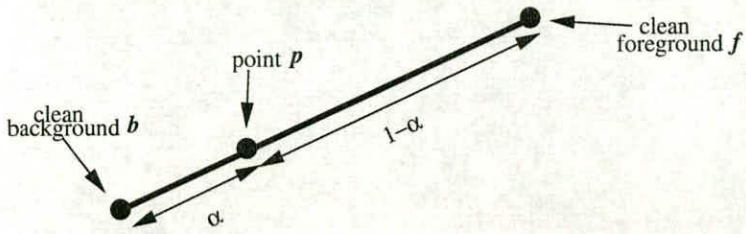


Figure 5.3: *Calculating α from clean colours: The α value of a point is the distance of the point along the normalised line between its clean foreground and background colours in colour space*

background or foreground. Instead, each pixel is given any value in the range $[0, 1]$. This is rather like using fuzzy logic [122], and assigning each pixel in the image a membership function of the foreground set. However, a defuzzification step to produce a crisp decision is not used, as the membership value of each pixel is the required result.

A technique for calculating alpha can be derived from the compositing equation by inverting it. Once a clean foreground $\boldsymbol{f}$ and an alpha value α have been calculated, they will be used to generate a pixel $\boldsymbol{p}$ with a different background $\boldsymbol{b}$ using $\boldsymbol{p} = \alpha\boldsymbol{f} + (1 - \alpha)\boldsymbol{b}$. Consider the input image to be produced by the same process. Pixels $\boldsymbol{p}$ in the input image can be treated as the result of a clean foreground colour $\boldsymbol{f}$ composited over a "clean background" colour $\boldsymbol{b}$ using an alpha value α. Segmenting the image consists of finding this alpha value α as well as the clean foreground $\boldsymbol{f}$ for every pixel in the image.

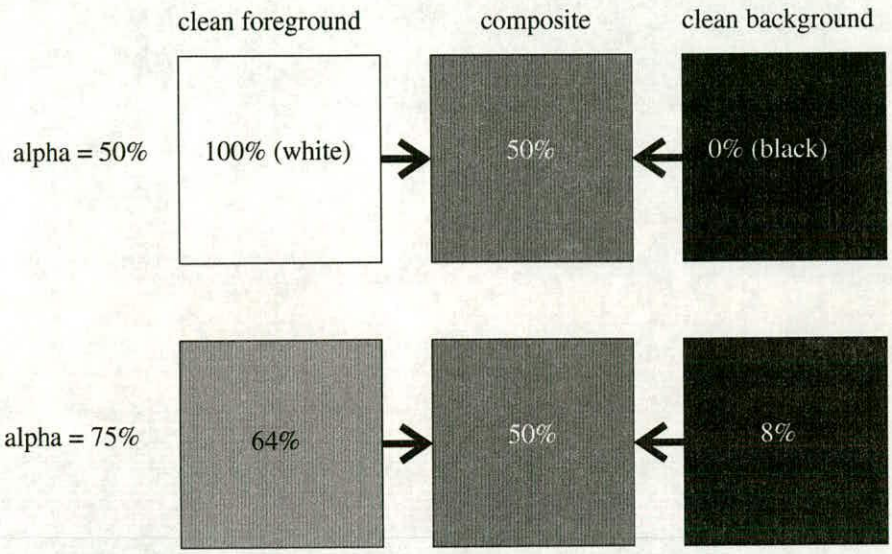


Figure 5.4: *To illustrate that the colour of a pixel is not enough information to determine the alpha value. A 50% grey pixel can be produced both by a 50% blend of black background and white foreground, and a 75% blend of two other shades of grey. Thus, the background and foreground colours must be known in order to find alpha*

α can be found by estimating $\boldsymbol{f}$ and $\boldsymbol{b}$: If colours are treated as vectors in 3d space, then $\boldsymbol{p}$ must lie on the line between $\boldsymbol{b}$ and $\boldsymbol{f}$, because it is formed by a linear combination of the two. α can be found using

$$\alpha = \frac{|\boldsymbol{p} - \boldsymbol{b}|}{|\boldsymbol{f} - \boldsymbol{b}|} \tag{5.1}$$

as shown in Fig. 5.3. This is in effect a rearrangement of the compositing equation (Equation 2.2).

Although there may be "true" clean background and foreground colours, they cannot be calculated exactly: There is no way of telling whether a pixel is 50% grey because it is a 50% mix of a clean black (0%) background and a clean white (100%) foreground, or a different mix of two other colours, as shown in Fig. 5.4. Smith and Blinn [3] proposed a solution that requires two different background colours, and eliminates this ambiguity. This extends Fig. 5.3 to be a triangle with two clean background colours and two pixel colours $\boldsymbol{p}$. However, using two backgrounds requires a static foreground, which is not possible for live actors, and also requires twice the amount of film, so this solution is far from ideal.

While there *may* be "true" clean background and foreground colours, there may not be. If

the foreground is truly transparent but also coloured (as in the case of tinted spectacles, or strands of hair) the alpha compositing equation does not hold. While Environment Matting [7] (see Section 2.3.1) will produce a correct composite, in this thesis an alpha channel and clean foreground colour are required. Hence there may not be a valid alpha and clean foreground colour. Since it is not possible to determine them precisely in any case, the best that is possible is to estimate a clean foreground colour and an alpha value that appear convincing when used together to composite over a new background.

Again, it is always possible to find an alpha value from a given pair of estimated foreground and background colours, given a pixel $\boldsymbol{p}$ situated on a line between clean colours $\boldsymbol{b}$ and $\boldsymbol{f}$. Because the clean colours may be inaccurate estimates, the point $\boldsymbol{p}$ may not lie directly on the line $\overrightarrow{\boldsymbol{fb}}$. Alpha can be defined to be the nearest point on the line $\overrightarrow{\boldsymbol{fb}}$ to the point $\boldsymbol{p}$, given by the equation

$$\alpha = \frac{(\boldsymbol{p} - \boldsymbol{b}) \cdot (\boldsymbol{f} - \boldsymbol{b})}{|\boldsymbol{f} - \boldsymbol{b}|^2} \tag{5.2}$$

and limiting the result to lie on the range $(0, 1)$. Essentially, this is estimating alpha from a point $\boldsymbol{q}$ which is the closest point to $\boldsymbol{p}$ on the line $\overrightarrow{\boldsymbol{bf}}$, as shown in Fig. 5.5. Equation 5.2 can also be expressed as:

$$\alpha = \frac{\left|\overrightarrow{\boldsymbol{bq}}\right|}{\left|\overrightarrow{\boldsymbol{bf}}\right|} \tag{5.3}$$

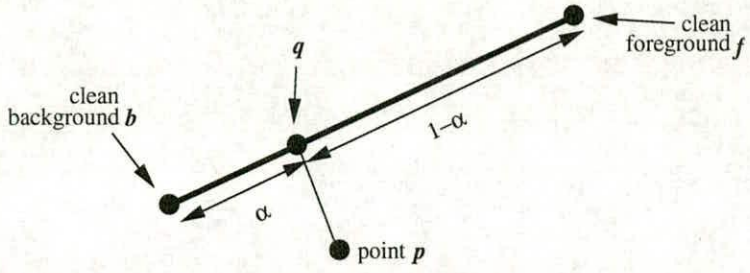


Figure 5.5: *Calculating α when the clean colours are estimates. Where $\boldsymbol{p}$ does not lie on $\boldsymbol{fb}$, the projection of $\boldsymbol{p}$ onto $\boldsymbol{fb}$ (point $\boldsymbol{q}$) is used to find α*

The distance from the point $\boldsymbol{p}$ to the line $\overrightarrow{\boldsymbol{fb}}$ can be used as a "confidence measure" to score the quality of the estimated foreground and background colours.

Therefore, an alpha value for every pixel in the unknown area can be generated by estimating the 'clean' foreground and background colours, and applying equation 5.2 to find the value of alpha.

This reduces the task of segmentation to that of estimating these clean colours (*i.e.* estimating what the background would look like without a foreground and *vice versa*). The different techniques presented here use different methods to estimate the foreground and background colours, and so estimate alpha.

5.4 Assessment of algorithms

The remainder of this chapter presents a sequence of progressively more complex algorithms which perform alpha estimation. The following criteria will be used to assess these algorithms:

- Relative runtime (computational complexity)
- Presence of large geometric artifacts in the alpha channel
- Presence of detail in the area of the clean foreground image that corresponds to the unknown area in the hint image
- Ability to resolve the hair at the top of the head in the *Gema* image

5.5 Nearest Pixel Estimation

A very simple way of estimating foreground and background is to set the estimated foreground and background colour for each pixel p within the unknown area to be the same colour as the physically closest definite foreground and background pixel to p, as shown in Fig. 5.6

This method produces results as shown in Fig. 5.7 for the *Gema* image. A noticeable feature of this is the geometric artifacts that result, caused by neighbouring pixels having a different nearest definite foreground point. Also, the estimated clean foreground has no detail for any pixel that lies within the unknown region — the colour of the pixels at the edge between the unknown region and the foreground are simply duplicated into the unknown area.

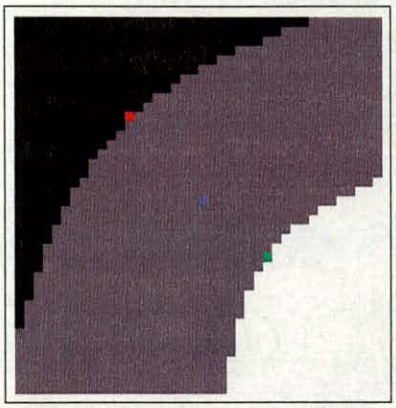


Figure 5.6: *Location of pixels used as clean colours in nearest pixel estimation. The point being classified is shown in blue. The background and foreground pixels used to classify the pixel are shown in red and green respectively*

5.6 Colour Correction

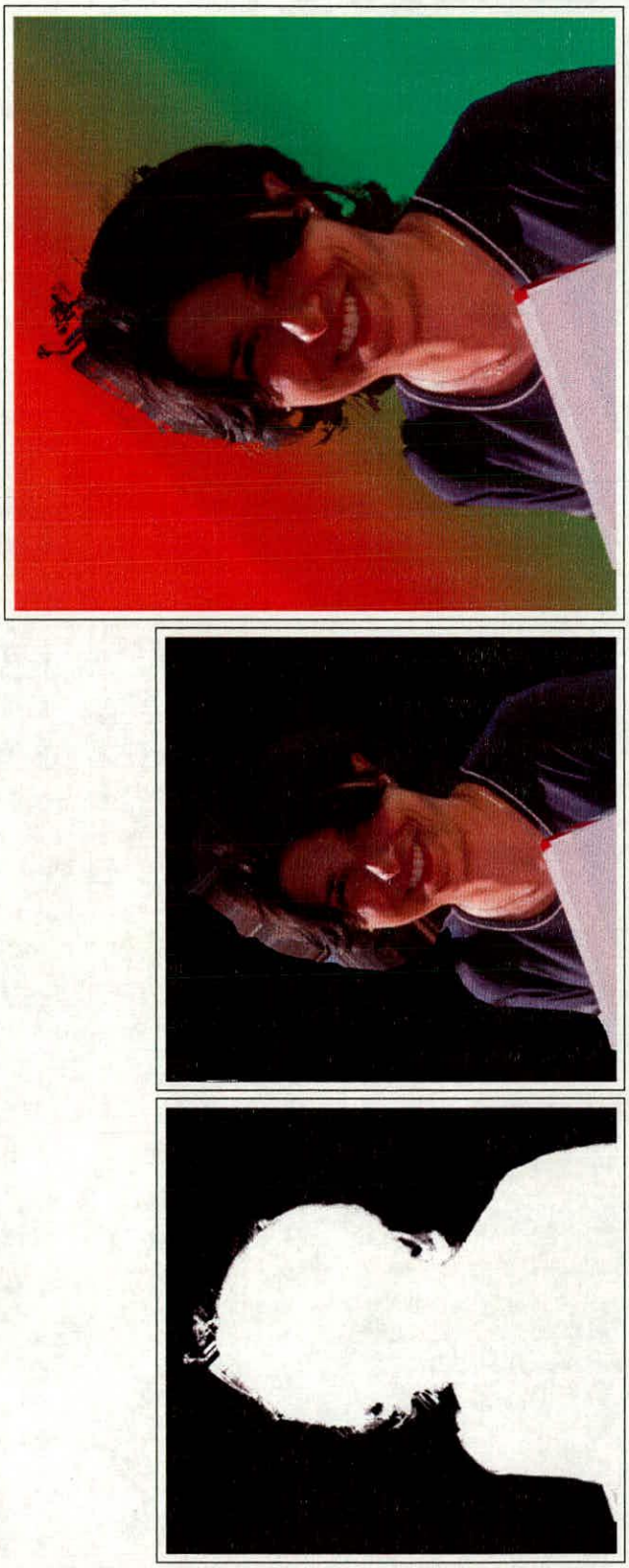
Setting the clean foreground colour to be *identical* to the nearest foreground colour causes the unknown area of the clean foreground image to become extremely uniform. Detail is lost in the unknown area. The following method can be used to recover this detail.

Fig. 5.8 shows the position of the estimated foreground $\boldsymbol{f}$ and background point $\boldsymbol{b}$ for a given pixel, in a 2d colour space normalised so that $|\boldsymbol{f} - \boldsymbol{b}| = 1$ for simplicity. Alpha is calculated by finding the nearest point $\boldsymbol{q}$ on the line $\overrightarrow{\boldsymbol{fb}}$ to the actual pixel colour $\boldsymbol{p}$. Alpha is then the ratio $|\overrightarrow{\boldsymbol{bq}}|/|\overrightarrow{\boldsymbol{bf}}|$. Recall that, according to the compositing equation 2.2, if the estimated background and foreground colour are correct, point $\boldsymbol{p}$ should lie on the line between $\boldsymbol{b}$ and $\boldsymbol{f}$. That is, $\boldsymbol{p}$ should be exactly where the point $\boldsymbol{q}$ is. One possible correction is to add the vector $\overrightarrow{\boldsymbol{qp}}$ to both the estimated foreground colour $\boldsymbol{f}$ and to the background colour $\boldsymbol{b}$, forming new points $\boldsymbol{f'}$ and $\boldsymbol{b'}$ respectively. $\boldsymbol{p}$ is now on the line $\overrightarrow{\boldsymbol{f'b'}}$ as required.

This correction scheme is used in all following algorithms and is also used in the work of other authors, except in that of Chuang *et al* [115].

5.7 Cluster-based Segmentation

Simply taking the nearest known background and foreground colours as the clean colours produces poor results because the clean colours may be inappropriate for the pixel being classified.



(c) Composite result

(b) Estimated Foreground

(a) Alpha channel

Figure 5.7: *Results of nearest pixel estimation technique*

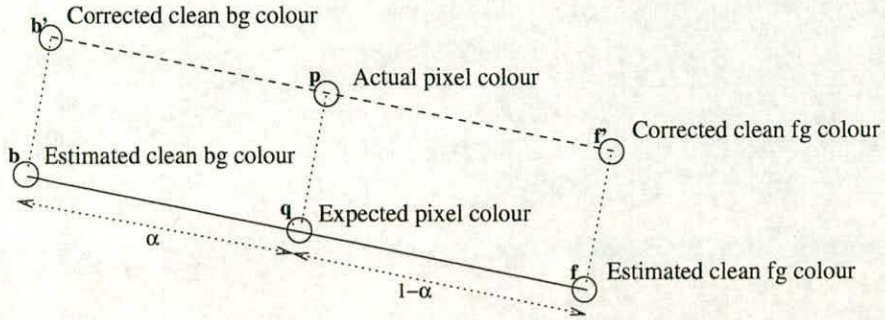


Figure 5.8: *Scheme for correcting estimated foreground colour*

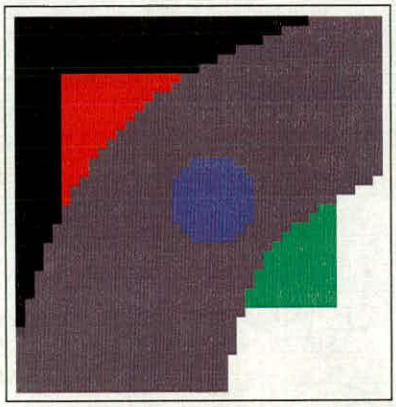


Figure 5.9: *Location of pixels used for clusters in the simple cluster based algorithm*

If either the background or foreground is non-uniform, the chance of this happening is greatly increased. One simple solution is to use a collection or *cluster* of nearby pixels, rather than just one. The clean colours are then derived from statistics of the cluster of pixel colours.

A simple algorithm to process a cluster of pixels is to form the background and foreground clusters out of the n nearest definite background and foreground pixels respectively, as shown in Fig. 5.9. The clean background and foreground colours can be set simply by averaging all pixels values in each cluster.

Searching for the nearest pixels to a pixel location p that are of a given type (in this case that are black or white in the hint image) is a time intensive process. A Voronoi diagram will indicate which foreground or background pixel is nearest to any given unknown pixel [123], but finding a list of the n nearest pixels is more time consuming. For simplicity, a simple approximation is to use a square spiral search pattern originated at p and increasing in radius until the requisite number of pixels have been selected. Since the average of the nearest n pixels will not change

significantly between neighbouring pixels, groups of pixels can be classified together, using the same clean colours for each.

Results of using this algorithm (with the colour correction scheme of Fig. 5.8) are shown in Fig. 5.10. There are noticeable artifacts in places caused by adjacent groups of pixels having different alpha values (note the staircase effect below her right ear) and several areas of missing foreground, where the foreground and background colours were not appropriate. The purple area by the right ear is due to a large discrepancy between the estimated clean foreground and background colours, and the actual colour of pixels in that area. This causes the corrected foreground colour $\boldsymbol{f'}$ to be very different from the clean foreground colour.

5.7.1 The Stripe Method of Selecting Pixels

It is possible that the nearest n pixels selected from the definite area may extend some distance back from the border between known and unknown areas in the hint image. This can create a problem when defining the edges. Fig. 5.11 shows a part of the *Gema* image with the hint image and the pixel clusters superimposed. The pixels being classified are coloured blue, and the nearest n foreground pixels shown in red. Some of the pixels included in this cluster are taken from an area which corresponds to skin, rather than hair, but the pixels being classified are all a mixture of background and hair. This problem can be avoided by reducing the number of pixels selected, and by redrawing the hint image. Reducing the number of pixels selected will make this technique susceptible to the same geometrical artifacts visible in the single nearest pixel algorithm, and redrawing the hint image may not always avoid the problem.

To form a cluster of pixels C close to an unknown pixel p:
find the k nearest fore/background edge pixels, and store in set C_1
foreach c in C_1:
  add all fore/background pixels within a fixed radius r of c to C
next c

Algorithm 3: *Method to form clusters of foreground of background points*

A way of ensuring the clusters used to form the clean foreground and background colours do not extend too far back into the known areas, but to still allow the clusters to contain enough pixels to reduce geometric artifacts is the *stripe method*, shown in Fig. 5.12 Edge detection is used to detect those pixels that form the boundary between the unknown and foreground area, and also the boundary between the unknown and background area. These are referred to as *foreground edge* and *background edge* pixels, respectively. Algorithm 3 defines the method for

(c) Composite result

(b) Estimated Foreground

(a) Alpha channel

Figure 5.10: *Results of using the cluster–based algorithm on the* Gema *test image*

Figure 5.11: *Problem caused by selection of pixels using algorithm of section 5.7*

Figure 5.12: *The* stripe method *of selecting pixels*

forming clusters of background and foreground pixels.

Fig. 5.13 shows the result of using the stripe method to select colours

5.7.2 Alpha range adjustment

Fig. 5.14 illustrates a problem that occurs when the variance of the foreground and background clusters is too high. Here, point $\boldsymbol{b}$ and $\boldsymbol{f}$ are the clean foreground colours being used to classify point $\boldsymbol{p}$. The point marked $\boldsymbol{q}$ is the projection of $\boldsymbol{p}$ onto the line $\overrightarrow{\boldsymbol{bf}}$. Alpha is set to be $\left|\overrightarrow{\boldsymbol{bq}}\right| / \left|\overrightarrow{\boldsymbol{bf}}\right|$, which in the case of Fig. 5.14 will be slightly less than 1. However, the colour of point $\boldsymbol{p}$ is very close to the colour of some of the points within the foreground cluster. Some points in the foreground cluster are in fact closer to the background cluster than $\boldsymbol{p}$, but will still be assigned an alpha of 1 because they are in the known region. Suppose a known foreground pixel $\boldsymbol{r}$ is on the edge of the foreground and unknown area, and is adjacent in the image to the unknown pixel $\boldsymbol{p}$, which may be identically coloured. Because $\boldsymbol{p}$ and $\boldsymbol{r}$ will be assigned different alpha values, there will be a visible edge in the image between the two, corresponding to the edge of the unknown area in the hint image. This is an undesirable artifact that should be removed.

(c) Composite result

(b) Estimated Foreground

(a) Alpha channel

Figure 5.13: *Results of using the* stripe *algorithm on the* Gema *test image*

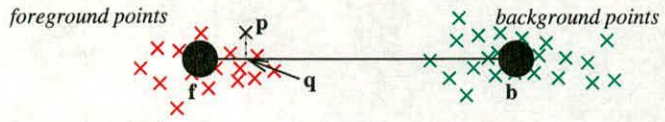


Figure 5.14: *Problem caused by high variance clusters*

A solution is to re-scale the alpha values. Normally, alpha is 0 at the centre of the background cluster and 1 at the centre of the foreground cluster. To remove artifacts when classifying a point $\boldsymbol{p}$, alpha can be rescaled, so that alpha is zero at the nearest point in the background cluster to $\boldsymbol{p}$, and alpha is one at the nearest point in the foreground cluster to $\boldsymbol{p}$. Now the point $\boldsymbol{p}$ in Fig. 5.14 will yield an alpha value of more than 1.0, which will be clipped to 1. The technique used to re-scale the alpha values is given in Algorithm 4.

find α using $\boldsymbol{b}, \boldsymbol{f}$ and $\boldsymbol{p}$ as usual.
let $\boldsymbol{n}_f$ be the nearest point in the foreground cluster to $\boldsymbol{p}$.
let $\boldsymbol{n}_b$ be the nearest point in the background cluster to $\boldsymbol{p}$.
let α_f be the classification of $\boldsymbol{n}_f$
let α_b be the classification of $\boldsymbol{n}_b$
set $\alpha \Leftarrow \frac{\alpha - \alpha_b}{\alpha_f - \alpha_b}$

Algorithm 4: *Alpha range adjustment*

If such a correction is applied, it follows that the (corrected) foreground and background colours should also be adjusted, so that compositing the clean foreground over the clean background exactly yields the original image. However, doing so will move these colours closer together, thus introducing traces of the background in the clean foreground image and *vice versa*. In this case, it is better to leave the clean foreground and background colours unaltered.

Alpha range adjustment is computationally intensive, and tends to produce a more contrasty alpha channel, as alpha values close to 0 and 1 are set to exactly 0 and 1 respectively. Alpha range adjustment should only be used where the edges in the hint image become visible.

5.8 Selective clean colour estimation

The next sections describe techniques that have a different basis. Instead of blindly estimating the foreground and background colours by extrapolation from known foreground and background colours, and then calculating alpha and correcting the estimates according to the actual colour, it is preferable to select clean foreground and background colours that are appropri-

ate for the colour of the pixel under classification. It may still be necessary to modify these estimates once alpha has been calculated.

5.9 Nearest Colour estimation

This novel algorithm is a simple "non-blind" method. To allow the algorithm to run faster, small neighbourhoods of pixels U from the unknown area are processed simultaneously, as in the nearest n define pixel algorithm of section 5.7. The known background and foreground pixels nearest to the centre of U in the image are selected to form two clusters B and F, respectively.

To process each pixel $\boldsymbol{p}$ in the set U, a clean background and foreground colour must be selected. Suppose the background cluster is a mixture of red and yellow, and pixel $\boldsymbol{p}$ is a shade of yellow. The simple cluster–based algorithm would average the cluster together, and give a clean background colour of orange. It is possible that this pixel is yellow because its clean foreground is green and its clean background is red (thus producing a yellow pixel) but it is perhaps more plausible that this pixel originates from the yellow part of the background. The Nearest Colour estimation algorithm is based upon the assumption that the best clean foreground or background colour to choose are those in the clusters F and B that are closest in colourspace to $\boldsymbol{p}$. Because the very closest pixel may be an outlier, the average of the K nearest colours are formed instead. The algorithm is described in Algorithm 5.

select a small region U of the unknown area
let F be the set of k_f pixels marked as foreground that are closest in the image to U
let B be the set of k_b pixels marked as background closest in the image to U
foreach pixel p in U
  let $\boldsymbol{p}$ be the colour of p
  let C_f be the n_f pixels in F that are closest in colourspace to $\boldsymbol{p}$
  let C_b be the n_b pixels in B that are closest in colourspace to $\boldsymbol{p}$
  set clean foreground colour $\boldsymbol{f}$ to be the mean of C_f
  set clean background colour $\boldsymbol{b}$ to be the mean of C_b
  set alpha using $\boldsymbol{f}$, $\boldsymbol{b}$ and $\boldsymbol{p}$ as before
next pixel p

Algorithm 5: *Nearest Colours algorithm to select clean foreground colours*

Alpha can now be calculated and the clean foreground and background colours corrected as described in the previous sections. The artifacts around the top of the head (where the clusters are overlapping) is increased in size — when the clusters overlap, it is often the case that the

(c) Composite result

(b) Estimated Foreground

(a) Alpha channel

Figure 5.15: *Results of using the Nearest Colours algorithm on the* Gema *test image*

clean background colour is equal to the clean foreground colour. However, the detail in the image (especially with the hair around the neck) is improved, as shown in Fig. 5.15.

5.10 Adaption with Principal Components Analysis

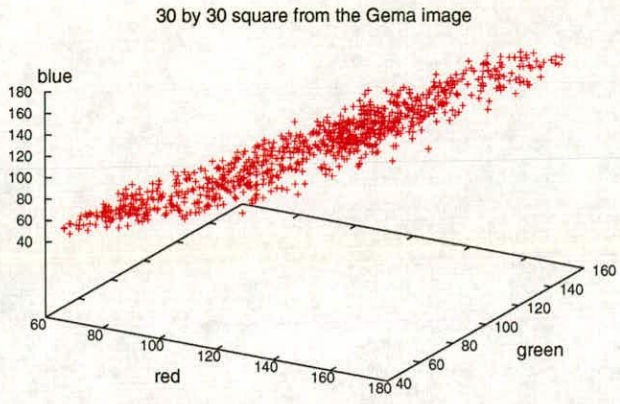


Figure 5.16: *Cluster of pixel colours from the* Gema *image, showing prolate shape (8 bit intensity values)*

Observation of many natural images has shown that pixels taken from small areas form **prolates** — they are cigar shaped in RGB space, as shown in Fig. 5.16. Ohta *et al* [72] made a similar observation: They took the Karhunen Loeve (PCA) transform of nine images, and measured the variance along all the PCA axes. They measured a variance along the major axis of approximately ten times that of the second and third axes, suggesting the clusters are prolates.

The following algorithm assumes that the foreground and background clusters are prolates in order to obtain extremely fast estimates of clean foreground and background colours.

The first step of this process finds the *principal axis*. The principal axis is given by the parametric line $\boldsymbol{s}(t) = \boldsymbol{\mu x} + \boldsymbol{r}t$, where $\boldsymbol{\mu}$ is the mean of the cluster and $\boldsymbol{r}$ is a vector orientated such that the mean squared distance of all points in the cluster to the line $\boldsymbol{s}(t)$ is minimised. $\boldsymbol{r}$ can be found using the Karhunen Loeve transform. Orchard and Bouman [20] show how to derive the principal components of a distribution in $\mathbb{R}^3$ for colour analysis:

Given a cluster C with n members $\boldsymbol{x}$, the mean $\boldsymbol{\mu}$ and covariance matrix $\bar{R}$ of C are found:

$$R = \sum_{s=1}^{n} \boldsymbol{x}_s \boldsymbol{x}_s^t \tag{5.4}$$

$$\boldsymbol{m} = \sum_{s=1}^{n} \boldsymbol{x}_s \tag{5.5}$$

$$\boldsymbol{\mu} = \frac{\boldsymbol{m}}{n} \tag{5.6}$$

$$\bar{R} = R - \frac{1}{n} \boldsymbol{m}\boldsymbol{m}^t \tag{5.7}$$

The vector $\boldsymbol{r}$ is given by the principal eigenvector $\boldsymbol{e}_0$ of $\bar{R}$ – *i.e.* that which has the largest corresponding eigenvalue λ. All eigenvectors and eigenvalues can be found using the `Jacobi` and `Eigsrt` routines from the Numerical Recipes collection [124]. The entire matrix of eigenvectors $E = \left[\boldsymbol{e}_0{}^t \boldsymbol{e}_1{}^t \boldsymbol{e}_2{}^t\right]$ (where $\boldsymbol{e}_i$ is the eigenvector with the i^{th} largest eigenvalue) is an orthogonal matrix that forms a transform into PCA space. That is, a new set C' with members $\boldsymbol{x'}_i = E\boldsymbol{x}_i$ is a set rotated about the origin with respect to C with the principal axis parallel to the x-axis, and the second and third aligned parallel to the y- and z-axes respectively.

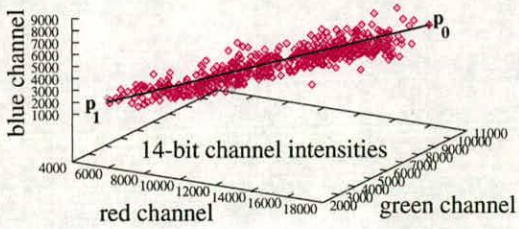


Figure 5.17: *Cluster of points in RGB space with the line* $\overline{\mathbf{p_0 p_1}}$

Once the set C' has been formed, the range (r_{min}, r_{max}) of the transformed set along the principal axis is found. The mean $[\mu_1, \mu_2, \mu_3]$ in each dimension of PCA space is also calculated. The transformed end points $\mathbf{p_{min}}$=$[r_{min}, \mu_2, \mu_3]$ and $\mathbf{p_{max}}$=$[r_{max}, \mu_2, \mu_3]$ are then inverse transformed back into RGB space to form $\mathbf{p_0}$ and $\mathbf{p_1}$. Fig. 5.17 shows a cluster, and the line $\overrightarrow{\mathbf{p_0 p_1}}$.

The range of the set is used rather than a statistical measure such as the standard deviation because the clusters are very rarely symmetric and do not fit well to simple statistical models. The range calculation is performed for both foreground and background, giving two lines $\overrightarrow{\mathbf{p_0 p_1}}_f$ and $\overrightarrow{\mathbf{p_0 p_1}}_b$, as shown in Fig. 5.18.

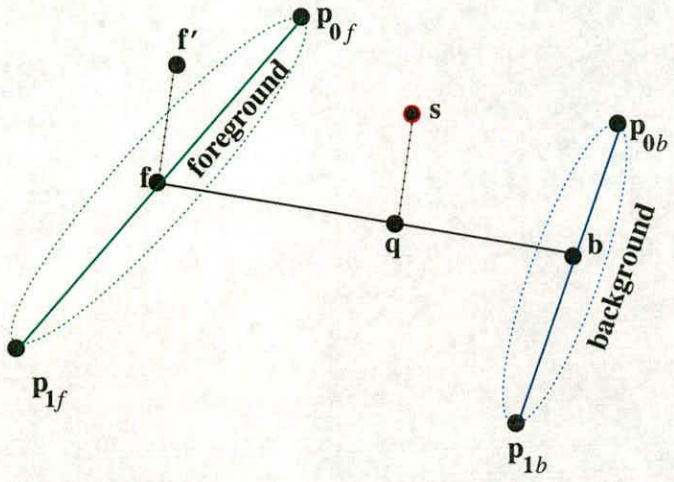


Figure 5.18: *Position of points used to classify in colour space*

It is now assumed that every pixel $\mathbf{s}$ in the set of unknown pixels U is composed of a clean background colour close to a colour $\mathbf{b}$, and a clean foreground close to a colour $\mathbf{f}$, where $\mathbf{b}$ is a point on the line $\overrightarrow{\mathbf{p_0 p_1}_b}$ and $\mathbf{f}$ lies on the line $\overrightarrow{\mathbf{p_0 p_1}_f}$. $\mathbf{f}$ and $\mathbf{b}$ are therefore the estimated clean foreground and background colours. The most appropriate points to choose for $\mathbf{b}$ and $\mathbf{f}$ are those points closest to $\mathbf{s}$, formed by finding

$$q = \frac{(\mathbf{s} - \mathbf{p_0}) \cdot (\mathbf{p_1} - \mathbf{p_0})}{|\mathbf{p_1} - \mathbf{p_0}|^2} \tag{5.8}$$

for foreground q_f and background q_b. If q_f and q_b are limited to the range $(0, 1)$, then

$$\mathbf{b} = \mathbf{p}_{1b} q_b + \mathbf{p}_{0b}(1 - q_b) \tag{5.9}$$

$$\mathbf{f} = \mathbf{p}_{1f} q_f + \mathbf{p}_{0f}(1 - q_f) \tag{5.10}$$

give points $\mathbf{f}$ and $\mathbf{b}$ constrained to lie between $\mathbf{p_0}$ and $\mathbf{p_1}$. These are then the clean foreground and background colours. The alpha value can then be calculated using

$$\alpha = \frac{(\mathbf{s} - \mathbf{b}) \cdot (\mathbf{f} - \mathbf{b})}{|\mathbf{f} - \mathbf{b}|^2} \tag{5.11}$$

again limiting the result to lie on the range $(0, 1)$. $\boldsymbol{f'}$, equal to $\boldsymbol{f} + \boldsymbol{s} - \boldsymbol{q}$, is the clean foreground colour as before.

Fig. 5.19 shows this algorithm applied to the *Gema* test image. There is no black artifact around the top left of the head, but there are many areas of noise. Most noticeably in the foreground, and in the alpha channel, there are small lines in the image. These occur because neighbouring pixels are classified using different foreground and background clusters. The clean foreground and background colours chosen can be significantly different for two neighbouring pixels, resulting in different estimates for alpha and the clean foreground colour, even if the pixels themselves are identically coloured.

5.11 Multiple Classification

The artifacts caused by neighbouring groups of pixels being classified using different foregrounds can be avoided in two ways. Either each pixel can be classified individually, or alternatively pixels can be classified multiple times in overlapping clusters, and the results averaged together. The latter tends to produce more acceptable alpha channels. The reclassification scheme is only necessary where the background and foreground are locally very varied.

To achieve this, every n^{th} row and column is examined for a pixel within the unclassified region, forming the set U out of all points within a radius r_c of the scanned pixel, whether or not they have been previously classified. Values of n and r_c are chosen such that a single pixel will be classified more than once. The final α value is calculated by forming a weighted average of the calculated α values. The weight for each classification is proportional to $\overrightarrow{\mathbf{fb}}/\overrightarrow{\mathbf{qs}}$ (all the weights used in classifying a single pixel sum to 1). This scheme favours cluster pairs positioned such that $\mathbf{s}$ lies close to the line $\overrightarrow{\mathbf{fb}}$, indicating that the $\mathbf{f}$ and $\mathbf{b}$ are likely clean foreground and background colours.

Fig. 5.20 shows that the multiple classification technique removes the artifacts.

5.12 Improvements using Region Growing

This section proposes two ways of increasing performance and speed of the PCA based technique of the previous section.

(c) Composite result

(b) Estimated Foreground

(a) Alpha channel

Figure 5.19: *Results of using the Principal Axis algorithm on the* Gema *test image*

(c) Composite result

(b) Estimated Foreground

(a) Alpha channel

Figure 5.20: *Multi-classification algorithm on the* Gema *test image*

5.12.1 Region Growing to Expand the Known Areas

Occasionally, the unknown area in the hint image is larger than it needs to be. This increases runtime, as more pixels need to be calculated. On fairly smooth images, the hint image can be made smaller automatically using a region growing approach. A separate region is grown using every point which is on the edge of the known and unknown areas as a seed pixel in turn. Candidate pixels in the unknown area are included in the region if they are adjacent to a pixel within the region, and the Euclidean distance between the colour of the candidate and the colour of the seed pixel is less than a threshold value. All of the pixels in a region that grew from a background edge seed are marked as background; all those that were grown from a foreground edge are marked as foreground. Thus, the unknown area is reduced in size.

The threshold must be chosen such that any pixel that should be assigned an alpha value other than zero or one will not be included in the region. In practice, a threshold is chosen that will terminate the region growing before any such pixel is reached.

5.12.2 Region Growing for Alpha Estimation

All algorithms presented here assume that both the background and foreground are uniformly coloured around the unknown area, so that clean foreground and background colours for pixels in the unknown area can be derived from pixels in the known foreground and background areas. However, it is also possible to sample clean foreground and background colours from previously calculated pixels within the unknown area. If the alpha value for an unknown pixel is calculated to be zero or one, then the clean background or foreground colour (respectively) will be identical to the pixel colour. If the pixel is any intermediate value, then a clean background and foreground colour will be updated (as in Fig. 5.8). These clean colours can be used to classify further pixels. Unlike previously described techniques, producing a clean background image is essential for this technique. This gives the advantage of improved accuracy, since the adjusted clean foreground samples f' for previously classified pixels can give better estimates for the clean foreground colour of the pixel under classification. This scheme also speeds up processing: Searching for nearby clean foreground and background pixels in order to form the foreground and background clusters takes longer if the pixels are further away. By allowing the re-use of nearby pixels, the search will finish quicker.

Because the values of some pixels within the unknown area will now depend on the results of

other pixels, some attention to the order in which the pixels are calculated must be considered. To ensure that each pixel has either a known foreground value or a known background value as an immediate neighbour, a region growing order can be employed. Pixels are taken alternately from nearby the foreground and background, and classified in turn.

let a *region growing queue* store for each item i:
    a pixel position i_{ij}
    the seed pixel colour $i_{\boldsymbol{p}}$
    whether the seed is background or foreground i_g

let an *alpha estimation queue* store for each item i:
    a pixel position i_{ij}

To estimate alpha:
push every edge point into region growing queue r
set alpha estimation queue p=GROW(r)
set alpha estimation queue w=ESTIMATE(p)

Function GROW with region growing queue r and threshold T:
set alpha estimation queue p to empty set $\emptyset$
while $r \neq \emptyset$
  pop next item i off r
  foreach neighbour n of i_{ij} that is still marked "unknown"
    let $\boldsymbol{n}$ be the colour of n
    if $\|\boldsymbol{n} - i_{\boldsymbol{p}}\| < T$
      if i_g is background, **mark** n_{ij} "background" **else mark** n_{ij} "foreground"
      push n onto region growing queue r, setting $n_{\boldsymbol{p}} = i_{\boldsymbol{p}}$
    else
      push n onto alpha estimation queue p
  next neighbour
return p

Algorithm 6: *Region growing approach to alpha estimation*

Algorithms 6 and 7 outline the process. The technique uses three functions: GROW expands the known areas and ESTIMATE produces clean foreground, clean background and alpha values for the remaining pixels in the unknown area by calling the CLASSIFY function. The algorithms do not classify pixels that fail the tests outlined, because the clean foreground and background colours generated are likely to be erroneous and it is important that the errors should not propagate to further pixels. Instead, these pixels are returned by the classifier function. It may be that once more pixels have been classified, a better estimate for background and foreground is known, in which case they will pass the classification. Thus, the queue w can be passed again to the ESTIMATE function. After several such iterations, the results of all classifications are accepted

Function ESTIMATE with region growing queue p
set alpha estimation queue q to $\emptyset$
while $p \neq \emptyset$
  pop next item i off p
  if i_{ij} marked as "unknown" or "failed"
    CLASSIFY point i_{ij}
    if classification was good
      set alpha, and clean background and foreground pixels
      mark pixel i_{ij} as "estimated"
    else
      push i onto q
      mark pixel i_{ij} as "failed"
    endif
    push all neighbours marked as "unknown" onto p
  endif
endwhile

To CLASSIFY a point

- Form clusters:
  - include any pixel marked as foreground or background in original hint image (if it is within radius r from the edge of known/unknown) with weight 3.
  - include any pixel marked as foreground or background using GROW with weight 2.
  - include the clean foreground or background colour of an estimated pixel (if the alpha is close to 1.0 or 0.0) with weight 1.
- Classify as in section 5.10, forming a weighted covariance matrix. Store alpha as well as clean background and foreground colours
- If alpha calculated from Equation 5.11 is too far outside range (0,1), or if the clusters too close together ($||\vec{\mathbf{bf}}||$ too short) the classification has failed

Algorithm 7: *Alpha estimation with the Region Growing approach*

no matter how poor.

There is an inherent uncertainty in using clean background and foreground colours that have been found by the growing or by the estimation process. For this reason, a weighting scheme is used. Equations 5.4 and 5.7 on page 66 then become:

$$R_n = \sum_{s=1}^{n} w_s \boldsymbol{x_s} \boldsymbol{x_s^t} \tag{5.12}$$

$$\bar{R} = R \frac{1}{n \sum_{s=1}^{n} w_s} \boldsymbol{m} \boldsymbol{m}^t \tag{5.13}$$

where w_s is the weight of $\boldsymbol{x_s}$.

Fig. 5.21 shows the result of applying the Region growing modification to the Principal Axis algorithm.

5.13 Backlighting

It is common that actors in scenes are illuminated from behind ('backlit'). This causes a highlight along the edge of the foreground, as shown in Fig. 5.22(a) (taken for the *Dragonheart* sequence). This helps the foreground to stand out (and therefore can help a composite to look more natural) but causes a problem when estimating alpha. Fig. 5.22(b) shows the alpha channel (in magenta) overlayed with the original area. The alpha channel is too small - the pixels that are highlighted seem to be classified as background rather than foreground.

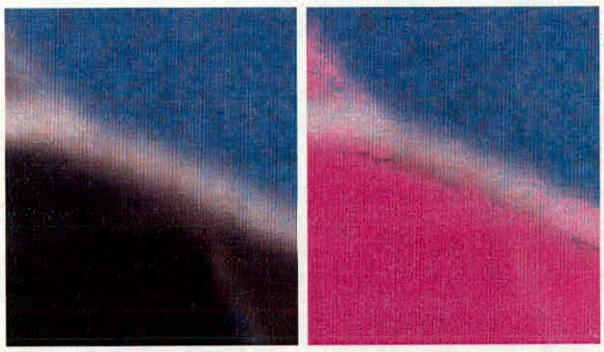
Fig. 5.23 plots the colour vectors for pixels in a column of Fig. 5.22(a), with the background and foreground ends marked in green and blue respectively. The red line connects consecutive pixels between these two end points. If Equation 5.1 is to hold, all the intermediate points should lie on a straight line between foreground and background. Instead, the points seem to diverge off to a white value as they cross the backlit area. The intermediate pixels are not a combination of the clean foreground colour found, but a combination of the three colours: The clean background, the clean foreground, and a "clean backlight" colour $\boldsymbol{m}$. Alpha estimation has produced background values for the backlit area because the background is brighter than the foreground, and the backlit area is brighter still. Thus, any backlit pixel will be classified as background instead of as foreground.

(c) Composite result

(b) Estimated Foreground

(a) Alpha channel

Figure 5.21: *Region growing algorithm on the* Gema *test image*



(a) Image showing effect of backlighting

(b) Bad matte produced by standard estimation

Figure 5.22: *Section of backlit image with matte overlayed (in magenta) showing poor alpha estimation*

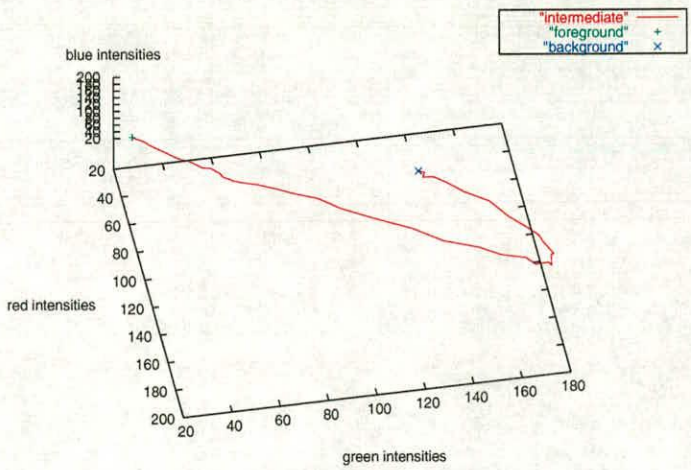


Figure 5.23: *Graph of pixel colours taken from a single column in the image of Fig. 5.22(a), showing the progression of colours from the foreground colour (left hand end) to the foreground (right hand end). The curve in the line is caused by backlighting*

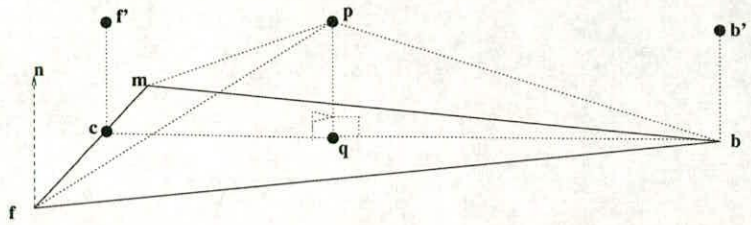


Figure 5.24: *Position of points used to classify in the presence of backlighting*

The backlighting problem can be corrected in this case by redrawing the hint image so that the unknown area covers only the transition between the backlit foreground and the background. However, this is not always possible (the backlighting might affect only a single hair or may only effect pixels that are a blend of background and foreground), so an alternative technique is proposed. This technique is only useful where the clean backlight colour is not co-linear with the clean foreground and background colours. If f, b and m are co-linear, there is no solution to the alpha estimation problem.

An extension of normal alpha values is required, which permits the mixture of three, rather than two, colours. Fishkin and Barsky [125] describe a technique on which this method is based. Given three clean colours f , b and m , we wish to know for each pixel p how much of each clean colour is required to produce p. Fig. 5.24 shows the position of these four points. Fishkin and Barsky [125] assume no noise; they assume that there is a linear combination of the clean colours that will perfectly reproduce p. This will only be the case if p lies on the plane defined by f , b and m. This will not be the case for real images under the presence of image noise. An alternative technique is used here. Rather than using point p directly, the point q is used, which is the nearest point from point p to the plane formed by f,b and m.

q is found by calculating the normal n to the plane:

$$n = (m - f) \times (b - f) \tag{5.14}$$

$$\overrightarrow{qp} = \frac{n}{|n|}\frac{(p - f) \cdot n}{|n|} \tag{5.15}$$

$$q = p - \overrightarrow{qp} \tag{5.16}$$

Now, the point c must be found. This is a point on the line $\overrightarrow{fm}$ that can be blended with b to

form the point $\boldsymbol{q}$. This is the point where the line from $\boldsymbol{b}$ through $\boldsymbol{q}$ intersects with $\overrightarrow{\boldsymbol{fm}}$. This is given by solving the simultaneous equations of the two lines:

$$\boldsymbol{c} = \boldsymbol{f} + x(\boldsymbol{m} - \boldsymbol{f}) \tag{5.17}$$

$$\boldsymbol{c} = \boldsymbol{b} + y(\boldsymbol{q} - \boldsymbol{b}) \tag{5.18}$$

Writing $\boldsymbol{u} = \boldsymbol{q} - \boldsymbol{b}$, $\boldsymbol{v} = \boldsymbol{b} - \boldsymbol{f}$ and $\boldsymbol{w} = \boldsymbol{m} - \boldsymbol{f}$ for short, and equating 5.17 and 5.18 gives

$$\boldsymbol{f} + x\boldsymbol{w} = \boldsymbol{b} + y\boldsymbol{u} \tag{5.19}$$

$$x\boldsymbol{w} - y\boldsymbol{u} = \boldsymbol{v} \tag{5.20}$$

$$x \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} - y \begin{bmatrix} u_0 \\ u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} v_0 \\ v_1 \\ v_2 \end{bmatrix} \tag{5.21}$$

This is an over-specified system, since it has two unknowns in three equations, and can be solved for x using just two of the three equations:

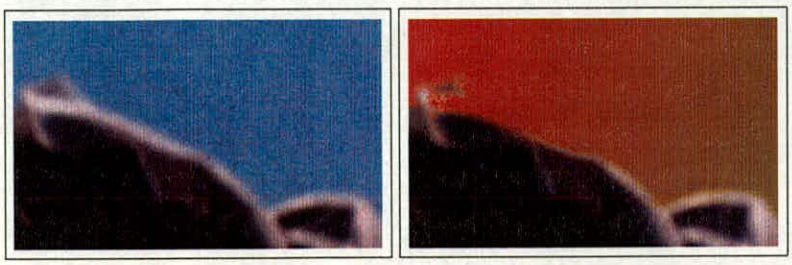
$$x = \frac{u_0 v_1 - u_1 v_0}{w_1 u_0 - w_0 u_1} \tag{5.22}$$

x can then be substituted into equation 5.17 in order to find $\boldsymbol{c}$. Alpha is found as normal:

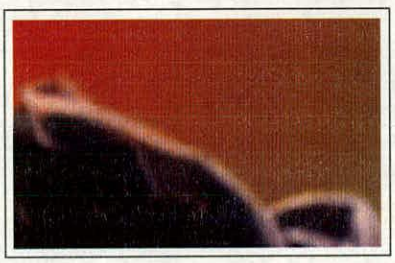
$$\alpha = \frac{|\boldsymbol{q} - \boldsymbol{b}|}{|\boldsymbol{c} - \boldsymbol{b}|} \tag{5.23}$$

Corrected clean foreground and background colours $\boldsymbol{f'}$ and $\boldsymbol{b'}$ are found by adding $\overrightarrow{\boldsymbol{qp}}$ to $\boldsymbol{c}$ and $\boldsymbol{b}$ respectively. Using α to blend these two colours exactly produces $\boldsymbol{p}$ as required.

If the region growing adaptation is used in order to collect foreground and background samples, an extra frame buffer is required with this system. The modified clean foreground colour $\boldsymbol{f} + \overrightarrow{\boldsymbol{qp}}$ is calculated and stored for use in forming the foreground clusters, but the value $\boldsymbol{f'}$ is output as the clean foreground image. Forming clusters out of clean foreground colours $\boldsymbol{f'}$ tends to



(a) Original image **(b)** Normal alpha estimation



(c) Backlight alpha estimation

Figure 5.25: *Result of using backlight alpha estimation*

produce noisy results, since the addition of pixels that are blends of f and m causes the clusters to cease to be prolate in shape.

5.13.1 Estimating the Clean Backlight Colour

Unlike the clean foreground and background colours, the clean backlight colour cannot be sampled directly from pixels in the image. There are unlikely to be any pixels that are purely backlit, with no trace of background, because backlighting tends to effect only the intermediate pixels. Even if a clean sample could be found, the backlight tends to be so bright that it would saturate the sensor.

It would be possible to estimate the "clean backlight" colour directly from the data. One technique might be to use a simple lighting model to derive the intensity of the backlight, given estimates of the other (ambient) light in the image and some estimate of the reflectance (and therefore colour) of the object. Lighting models are discussed by Foley *et al* [19].

The backlight colour can also be estimated geometrically from the pixel values. It will be a

point m such that f , b and m enclose virtually every point within a small area of the transition region. Solving this problem is similar to calculating the convex hull of the points [126]. In this case, it is necessary to find the best fit plane P to the surfaces, and then find a single triangle on the plane P that encloses the projection of all points onto P.

However, since the clean foreground colour tends to a uniform colour and intensity across the whole image, it can be easily estimated and entered by hand.

Fig. 5.25 shows the result of using the backlight alpha estimation algorithm on a small area of a frame scanned from the Motion Picture *Dragonheart*. The backlight colour in this case was set to [512,512,512], twice the intensity of full brightness in the frame.

5.14 Summary

This chapter has outlined the development of a segmentation algorithm that generates an alpha channel and clean foreground image from a source image. It requires a *hint image*, which indicates the approximate location of the boundary between the foreground and the unwanted background. Several algorithms have been proposed. All of these work by estimating a clean foreground and background colour, and then generating an alpha value by comparing the colour of the pixel to the colours of the clean foreground and background colours. The most complex — and most accurate — of these techniques is the Principal Axis technique, which generates clean foreground and background colours by sampling nearby clean foreground and background colours and approximating these distributions as straight lines.

Table 5.1 illustrates the relative performance of the algorithms.

The Principal Axis algorithm with region growing is the fastest algorithm and produces no geometric artifacts in this test image. In other images tested, the Principal Axis algorithm always outperformed the other algorithms. While it would be possible to derive a test image on which the Principal Axis algorithm performed more poorly than the others, suitable selection of parameters can be used to solve the problem. For example, setting the number of foreground and background pixels to sample to be one single pixel, and setting the region growing limits to zero will produce an identical result to the Nearest Pixel algorithm.

Since the Principal Axis algorithm outperforms the other algorithms, it is the algorithm of choice and will be the only one presented in this chapter which will be used in the remainder

Algorithm	**Relative Runtime**	**Artifact free**	**Detail in clean foreground**	**Hair resolved**
Nearest Pixel	2	No	No	No
Simple Cluster–based	3	No	Some	Partly
Stripe–based	3	No	Some	Partly
Nearest Colour	5	Partly	Yes	Partly
Principal Axis	4	Yes	Yes	Partly
Principal Axis (multi-classification)	8	Yes	Yes	Almost
Principal Axis (region growing)	1	Yes	Yes	Yes

Table 5.1: *Relative performance of the algorithms presented in Chapter 5*

of this thesis.

An adaptation to the algorithm to deal with backlighting has also been proposed.

The results presented in this chapter have all been based on the *Gema* image to allow comparison of the different methods. However, many other images have also been tested. Some of these results are presented in the next chapter where comparison with recent work by other authors in the work of alpha channel estimation will also be given.

Chapter 6
Comparison with Alternative Techniques

6.1 Introduction

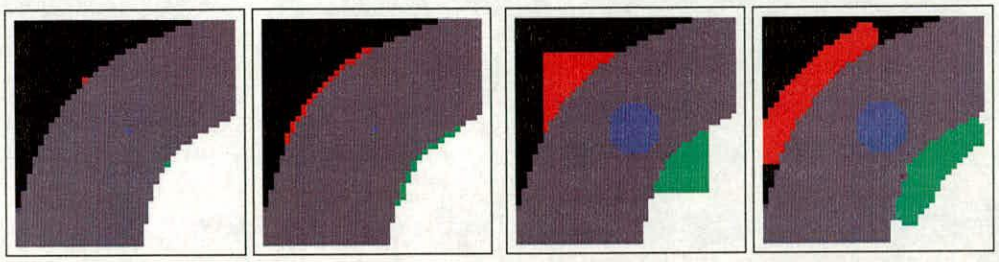
The algorithms described in the previous chapter were developed independently. However, during the period of research for this thesis, various algorithms for image segmentation were published that perform in a similar way.

This chapter presents these algorithms and compares them to the novel algorithms presented in the previous chapter. Two of these competitor algorithms are based around the use of Gaussian Mixture Models (GMMs) to represent colour distributions. These algorithms have inspired the development of a novel algorithm, which is described later in the chapter.

This chapter is structured as follows:

Section 6.2 describes the *KnockOut* algorithm. Section 6.3 presents the first of the GMM techniques, published by Ruzon and Tomasi [114]. The second GMM technique, published by Chuang *et al* [115] is described in Section 6.4. Chuang *et al* consider alpha estimation to be a Bayesian problem, and simplify the problem in order to produce timely solutions. Section 6.5 describes a novel algorithm that uses a Genetic Algorithm to solve this Bayesian problem without such simplification. Section 6.6 presents results of various alpha channel estimation algorithms. The chapter is summarised in Section 6.7.

This chapter concludes with a comparison of alpha estimation techniques, comparing the work of other authors with the algorithms outlined in the previous chapter with the new algorithm proposed in this chapter.



(a) Nearest 1 pixel (b) KnockOut (c) Nearest n pixel (d) Slice

Figure 6.1: *Different methods for selecting clusters of foreground and background pixels*

6.2 The *KnockOut* Algorithm

The algorithm used by the commercial software package Corel *KnockOut*[1] [116, 127] avoids the streaking problem caused by selecting the nearest known pixel, and tends to retain more detail.

Instead of setting the estimated foreground and background colours for each pixel p in the unknown area to be the same as the colour of the nearest known pixel, they are set as a weighted average of the colours of nearby known pixels, where the weighting is inversely proportional to the physical distance. This approach to selecting pixels is compared to the previous algorithms in Fig. 6.1. This averaging technique blurs out the streaking effect seen in Fig. 5.7. The blurring will be most effective if the background is itself blurred or uniform.

A variation on equation 5.2 is used to calculate a value for alpha. Instead of using the dot product, the following equation is used:

$$\alpha = \frac{q(\boldsymbol{p}) - q(\boldsymbol{b})}{q(\boldsymbol{f}) - q(\boldsymbol{b})} \tag{6.1}$$

where $q()$ is a function that returns a single 'channel' from the pixel (*i.e.* returns a scalar from the colour vector). Six functions $q()$ are used: R,G,B,B-G,B-R,G-R. These are weighted together according to the value of the denominator of the above equation, which measures how separated the background and foreground colours are in this projection of colourspace. Large

[1] *KnockOut* was originally developed by Ultimatte, who are the assignees of the patent

values of $q(\boldsymbol{f}) - q(\boldsymbol{b})$ will give less error in the alpha value, and are therefore more heavily weighted. This technique is akin to finding the linear transform of the vectors into a new colour space that best separates the colour. What improvement this provides (in the general case) over using the vector distance in RGB space (Equation. 5.2) is unclear.

The *KnockOut* patent [127] lists three possible schemes for calculating a corrected colour, one of which is identical to the method shown in Fig. 5.8.

6.3 Ruzon and Tomasi's Technique

Ruzon and Tomasi [114] adopt a clean foreground and background selection algorithm based on mixture models. Their clusters are collected in an approach similar to the nearest n method shown in Fig. 6.1(c) except that they collect all pixels within a square area based around the group of pixels to be classified. The clusters are then quantised into subclusters using Orchard Bouman's Colour Quantisation Technique [20]. The quantisation produces subclusters

$$\begin{aligned} F &= \{(\boldsymbol{v_1}, \sigma_{\boldsymbol{v}_1}, x_1), (\boldsymbol{v_2}, \sigma_{\boldsymbol{v}_2}, x_2), \ldots, (\boldsymbol{v_n}, \sigma_{\boldsymbol{v}_n}, x_n)\} \\ B &= \{(\boldsymbol{w}_1, \sigma_{\boldsymbol{w}_1}, y_1), (\boldsymbol{w}_2, \sigma_{\boldsymbol{w}_2}, y_2), \ldots, (\boldsymbol{w}_m, \sigma_{\boldsymbol{w}_m}, y_m)\} \end{aligned}$$

where $\boldsymbol{v}$ and $\boldsymbol{w}$ are the subcluster means, σ are their variances, and x_i and y_i are the proportion of points within the super-cluster allocated to subcluster i. Ruzon and Tomasi now assume that the clean foreground colour is some linear combination of the foreground subcluster means, and calculate the colour and α by examining pairs of subclusters in foreground and background.

Connecting lines are constructed between background and foreground clusters. Lines that nearly cross, or start from the same cluster but have a narrow angle between them, are rejected as these cause spurious values. Along each line, alpha is one at the foreground end and zero at the background end. At each alpha value along the line, a probability distribution is generated by interpolating the mean and the variance of the distributions at the ends of the line. Thus, the probability for given unknown pixel p at a particular alpha value α can be found by summing the probability of p for each line, using α to interpolate between the mean and variances at the end points.

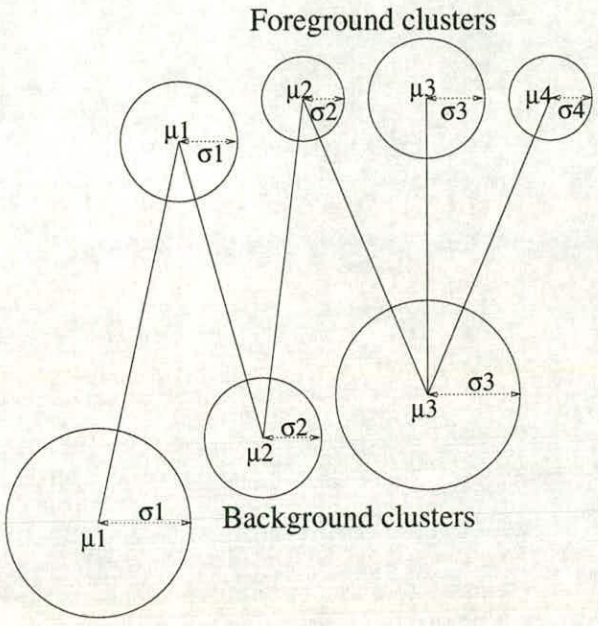


Figure 6.2: *Model used by Ruzon and Tomasi*

The α value chosen is the one that yields the highest total probability. Since alpha channels tend to be 8-bit, all 256 possible alpha values are tested separately. Once the best alpha is known, the clean foreground colour is formed using a linear combination of foreground subcluster means. The weight of each subcluster is proportional to the total probability of all lines starting at that cluster, calculated for the final value of alpha. Fig. 6.2 shows the arrangement of subclusters with interconnected lines used by the algorithm. The correction of Fig. 5.8 is applied to this colour to provide an accurate clean foreground colour.

This algorithm is equivalent to the "mean of nearest pixels" algorithm of section 5.7 if the variance is assumed to be constant on each side, and there is only one subcluster for each of foreground and background. For a given alpha value, the alpha-weighted mean of foreground and background is chosen. This provides a possible value of the point $\boldsymbol{q}$ in Fig. 5.8. If the variance was constant along the line, the final value of alpha chosen would be the one that produces a value for $\boldsymbol{q}$ closest in colour space to $\boldsymbol{p}$, the colour of the pixel being classified. Ruzon and Tomasi's algorithm is in effect an extension of the mean of nearest pixels algorithm, with the addition of variance and multiple clusters.

Because only a single variance measure is used, Ruzon and Tomasi assume that the clusters are spherical Gaussian distributions. That is, the covariance matrix is a scalar multiple of the

identity matrix: the variance is the same in every direction. If the distribution of foreground points is a thin prolate, as is frequently observed, this model will be inaccurate.

6.4 The Technique of Chuang *et al*

Chuang *et al* [115] also use mixture models, but use a Bayesian approach to estimate the clean foreground and background colours.

Given two distributions, modelled as mixtures of Gaussians, the problem of finding the clean foreground, clean background and alpha value is formulated using Bayes' rule:

$$\arg\max_{\boldsymbol{f},\boldsymbol{b},\alpha} P(\boldsymbol{f},\boldsymbol{b},\alpha|\boldsymbol{p}) = \arg\max_{\boldsymbol{f},\boldsymbol{b},\alpha} \frac{P(\boldsymbol{p}|\boldsymbol{f},\boldsymbol{b},\alpha)P(\boldsymbol{f})P(\boldsymbol{b})P(\alpha)}{P(\boldsymbol{p})} \quad (6.2)$$

$$= \arg\max_{\boldsymbol{f},\boldsymbol{b},\alpha} L(\boldsymbol{p}|\boldsymbol{f},\boldsymbol{b},\alpha) + L(\boldsymbol{f}) + L(\boldsymbol{b}) + L(\alpha) \quad (6.3)$$

where $P(\boldsymbol{b})$ represents the likelihood associated with clean background estimate $\boldsymbol{b}$ given a background distribution B, $L()$ denotes the log likelihood, and the $-L(\boldsymbol{p})$ term can be dropped because it is constant throughout optimisation. The first term of Equation 6.3 is derived from the compositing equation:

$$L(\boldsymbol{p}|\boldsymbol{f},\boldsymbol{b},\alpha) = -\frac{\|\boldsymbol{p} - \alpha\boldsymbol{f} - (1-\alpha)\boldsymbol{b}\|^2}{\alpha_{\boldsymbol{p}}^2} \quad (6.4)$$

This models the difference between the colour observed and the colour obtained by combining alpha and the estimated clean background and foreground colours as a Gaussian distribution with variance $\alpha_{\boldsymbol{p}}^2$. In order to estimate $L(\boldsymbol{f})$, the nearby foreground points are partitioned into several clusters, as for Ruzon and Tomasi. For each subcluster, the mean $\overline{f}$ and covariance matrix Σ_F are found. $L(\boldsymbol{f})$ is then given by

$$L(\boldsymbol{f}) = -\frac{(\boldsymbol{f} - \overline{\boldsymbol{f}})^T \Sigma_{\boldsymbol{f}}^{-1} (\boldsymbol{f} - \overline{\boldsymbol{f}})}{2} \quad (6.5)$$

Where the background is non–uniform, $L(\boldsymbol{b})$ is calculated in the same manner. The maximum likelihood values for alpha and the clean colours are found by iteratively estimating each one in turn. Assuming α is constant allows estimation of clean foreground and background. The maximum is found by differentiating Equation 6.3, setting the result to zero and solving a 6×6 linear equation:

$$\begin{bmatrix} \Sigma_F^{-1} + I\frac{\alpha^2}{\alpha_{\boldsymbol{p}}^2} & I\frac{\alpha(1-\alpha)}{\alpha_{\boldsymbol{p}}^2} \\ I\frac{\alpha(1-\alpha)}{\alpha_{\boldsymbol{p}}^2} & \Sigma_B^{-1} + I\frac{\alpha^2}{\alpha_{\boldsymbol{p}}^2} \end{bmatrix} \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{b} \end{bmatrix} = \begin{bmatrix} \Sigma_F^{-1}\overline{\boldsymbol{f}} + \boldsymbol{p}\frac{\alpha^2}{\alpha_{\boldsymbol{p}}^2} \\ \Sigma_B^{-1}\overline{\boldsymbol{b}} + \boldsymbol{p}\frac{1-\alpha^2}{\alpha_{\boldsymbol{p}}^2} \end{bmatrix} \tag{6.6}$$

where I is the 3×3 identity matrix. Each entry in the leftmost matrix is therefore itself a 3×3 matrix.

Once an estimate for the clean colours has been found, alpha is found using the inverted compositing equation (Equation 5.2). This alpha value can be used again to estimate new clean background and foreground colours.

Each cluster pair is used independently, and the values chosen are the pair that produce the highest maximum likelihood.

This technique models the subclusters with covariance matrices rather than using a single value of variance as Ruzon and Tomasi do. The subclusters are therefore treated as orientated elliptical Gaussian distributions, rather than as spherical Gaussian distributions.

The computational cost of both mixture model algorithms is fairly similar. Both techniques require the use of Orchard-Bouman's quantisation algorithm [20], that requires several passes through the supercluster, solving a 3×3 system of linear equations each time in order to extract the principal axis. Ruzon and Tomasi then exhaustively calculate multiple probabilities at each step for each possible alpha, while Chuang *et al* solve a 6×6 system of linear equations several times for each cluster pair.

Chuang *et al* have a slightly different method of selecting clusters when classifying pixels. As well as selecting pixels from the known foreground and background areas of the hint image, previously calculated clean foreground and background colours in the unknown area are used to form the clusters. To avoid artifacts caused by classifying pixels in groups, the clusters are formed separately for each pixel.

6.5 A Genetic Algorithm Solution to Bayesian Matting

Chuang *et al* make several significant assumptions in their algorithm:

1. The optimal alpha value can be found by examining each subcluster pair in turn and selecting the values for the pair which gives the largest value for equation 6.3. This is effective if the foreground and background clusters are multi-modal (so that there is no significant overlap between any two subclusters) but if the distributions overlap the maximum of Equation 6.3 may be missed by this technique.

2. The technique of starting with a mean value for alpha and then iteratively solving for the best values of $\mathbf{f}$ and $\mathbf{b}$ will converge to maximise Equation 6.3 for the given pair of clusters. Chuang *et al* do not formally demonstrate that this is the case.

3. All alpha values are equally likely. Suppose a pixel to be classified is surrounded by pixels previously classified as 100% foreground. If the pixel under classification is the same colour as these neighbouring foreground pixels, it is reasonable to assume that the pixel is more likely to be foreground than background, because images tend to be *spatially coherent*. Thus, the term $L(\alpha)$ in Equation 6.3 is not constant.

It is possible to circumvent these problems by attempting to find the solution that maximises Equation 6.3 across the entire cluster (rather than individual cluster pairs) while also allowing for an individual variation in the probability of alpha values (using the $L(\alpha)$ term). The term $L(\boldsymbol{f})$ now becomes

$$L(\boldsymbol{f}) = \log\left(\sum_{j=0}^{N_f} \left|\Sigma_{F_j}^{-1}\right| e^{-\frac{(\boldsymbol{f}-\overline{\boldsymbol{f}_j})^T \Sigma_{\boldsymbol{f}_j}^{-1} (\boldsymbol{f}-\overline{\boldsymbol{f}_j})}{2}}\right) \tag{6.7}$$

where N_f is the number of foreground subclusters, and $|M|$ indicates the determinant of the matrix M.

6.5.1 Modelling Spatial Coherency Using $L(\alpha)$

$L(\alpha)$ can be used to model spatial coherence by sampling nearby alpha values, including those of known background and foreground pixels (in which cases alpha is 0 and 1 respectively).

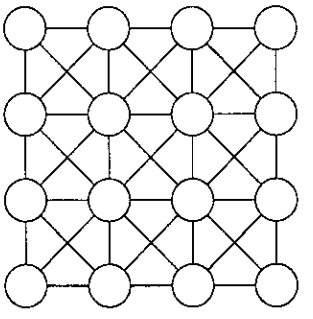


Figure 6.3: *Treating an image as a graph: Each vertex (shown as a circle) is a pixel, connected by eight edges to its neighbours.*

$$L(\alpha) = \log\left(\sum_{j=0}^{N_\alpha} \frac{1}{\sigma_{\alpha_j}} e^{-\frac{\alpha-\alpha_j}{2\sigma_{\alpha_j}}}\right) \tag{6.8}$$

where N_α is the number of pixels sampled and α_j is the alpha value of the j^{th} pixel sampled. σ_{α_j} is a variance associated with the j^{th} pixel and can be calculated in various ways: It can be made to be proportional to the physical distance in the image between the j^{th} pixel and the pixel under classification, or to the difference in colour between the two pixels in the input image. It is also possible to measure the "cost–of–path" distance using graph theory. Consider each pixel to be a node of a graph, as shown in Fig. 6.3.

Each pixel is considered to be connected to its neighbouring pixels by an edge with a weight or cost equal to the colourspace distance between the pixel colours in the input image. Thus, if there is a physical edge in the input between the two pixels, the edge connecting them will have a high cost. There are multiple paths between any two pixels in the image, each with a different total weight or cost (equal to the total weight of edges on the path between the intermediate pixels). The path that has the minimum total cost can be found by an algorithm due to Dijkstra [128, 129]. The weight of the minimum cost path between the pixel under classification and the pixel for which alpha has been sampled is a good choice for variance because it is proportional to both the distance between pixels on the image and the colour difference between them. It also has a natural tendency to enforce spatial coherence. Finding the minimum cost path between one starting pixel and every pixel in the image is very similar to region growing, except that there is no termination. Every pixel in the graph is considered and

assigned a cost, which is equivalent to the fuzzy membership of the graph. Using the minimum cost path as a term for σ_{α_j} thus has a tendency to encourage a pixel that is in the same region as α_j to be assigned a similar alpha value. Since the total cost of the minimum cost path is small for two pixels that are close in colour and distance, the variance will be small. Since Dijkstra's algorithm is fairly computationally intensive, it may be favourable to use just the spatial or colour difference between pixels.

6.5.2 Optimisation of the Bayesian problem

Chuang *et al* simplified the Bayesian formula to a system which had a single maximum. This maximum could be found by differentiating the system and finding the zero crossing point. If no such simplification is used, there are multiple maxima, potentially many hundreds. Finding the overall maximum by differentiation is no longer possible and a numerical optimisation strategy must be used instead. There are several algorithms that search for maxima in functions, many of which are described in [124]. Genetic Algorithms, described in the next section, have been adopted for this research partly due to their adaptability but also because there is much local expertise in their implementation.

6.5.3 Genetic Algorithms

When Equation 6.3 is applied to single Gaussian clusters without a term in $L(\alpha)$, there is only a single maximum that can be found by taking partial derivatives, setting them to zero and solving the resulting set of linear equations. However, where $L(F)$ and $L(B)$ are taken over mixtures of Gaussians and also a term in $L(\alpha)$, the derivatives can no longer be solved as a linear solution. However, the maximum can be approximated using a Genetic Algorithm. Genetic Algorithms are considered in detail by Goldberg [130]. Mimicking natural evolution, GAs attempt to solve a solution by having a *population* of candidate solutions, or *chromosomes*. Each chromosome has a *fitness*. Over several iterations, or *generations*, some inefficient solutions are discarded, and new solutions are made either by *mutation* (applying a random modification to one of the parameters within a randomly selected chromosome), or by *crossover* (combining parts of two randomly selected chromosomes). The best solutions to the problem — the *elites* — are passed unaltered to the next population. At the end of the run, the chromosome with the highest fitness is used as the final solution. GAs are used in a variety of search and optimisation problems, for example in the automatic generation of electronic circuits. [131].

Genetic Algorithms are used here to search for a near-optimal solution to the Bayesian problem (Equation 6.3). Raymer *et al* [132] also used a GA to optimise a Bayesian problem. In the case of solving the Bayesian Matting problem, the chromosome consists of a candidate clean foreground colour $\boldsymbol{f}$ and $\boldsymbol{b}$, and a candidate alpha value. Although Genetic Algorithms are generally initialised with a population of randomly selected individuals, a solution must be found quickly. Thus, part of the initial population is set with $\boldsymbol{f}$ and $\boldsymbol{b}$ equal to the means of each subcluster (with alpha calculated using Equation 5.2) so that the population tends to contain fit chromosomes before it begins.

Algorithm 8 outlines the process.

To find a good solution to Equation 6.3 using a Genetic Algorithm:

- Each chromosome has $\boldsymbol{f}$ and $\boldsymbol{b}$ and α : 7 values in total
- Initialise foreground and background clusters as before and subdivide into sets of sub-clusters F and B respectively

for r = each foreground cluster in F
  for s = each background cluster in B
    Initialise next chromosome in population P with $\boldsymbol{f} = F_{\mu_r}$ $\boldsymbol{b} = B_{\mu_s}$
    calculate α from equation 5.2
  next s
next s
set remaining population to be MUTATIONS of first $m \times n$ members
for g generations
  calculate fitness for each chromosome using equation 6.3
  create empty population Q
  add e best solutions from P into Q
  add m MUTANT solutions to Q
  add c CROSSOVER solutions to Q
  set $P \Leftarrow Q$
next generation
return best chromosome

To create a MUTANT
let c be the result of a TOURNAMENT
modify one of the seven values of c by a small random amount (proportional to the standard deviation of the cluster)
return modified c

To create a CROSSOVER
let c_0 and c_1 be results of two TOURNAMENTS
let d be a chromosome
for $a \in [0, 7]$
  let r be a 0 or 1 at random
  set $d[a] = c_r[a]$
next a
return d

To run a TOURNAMENT
select at random two members c_0, c_1 of population P
if $\text{fitness}(c_0) > \text{fitness}(c_1)$ **return** c_0 **else return** c_1

Algorithm 8: *Genetic Algorithm to solve Bayesian Matting*

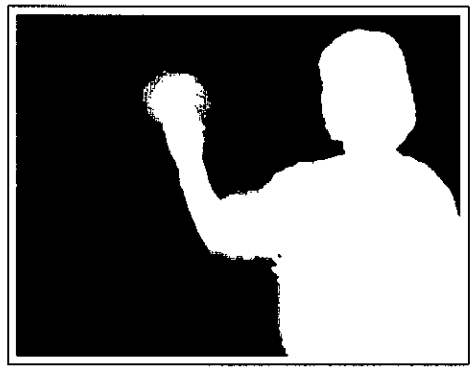


Figure 6.4: *Hand-generated ground truth alpha channel used to quantify results for frame 30 of the* Rachael *test sequence*

6.6 Results

6.6.1 Quantitive Results

Assessing the quality of an image segmentation method is a difficult process. There are no accepted techniques for such an analysis. One possible method would be to generate an artificial alpha channel and use it composite two different images together, then test the segmentation algorithm by running it on the composited image and attempting to recover the original alpha channel use the segmentation algorithm under test. The alpha channel produced by the segmentation algorithm can then be compared to the artificially generated "Ground Truth" alpha channel. Such artificial images lack many of the characteristics of genuine images — it is very difficult, for example, to match the lighting characteristics between the two images and to mimic effects such as backlighting and motion blur. Thus, an algorithm that performs comparatively poorly on this artificial image may perform significantly better on genuine images, and *vice versa.*

An alternative technique is to perform a hand segmentation of an image, and to use this alpha channel as Ground Truth. In the general case, this is an extremely laborious and difficult process. For the purpose of this thesis, a bluescreen image was used for testing. An alpha channel (Fig. 6.4) was generated for this image by combining the alpha channel produced by CFC's *Keylight* package with other alpha channels, and modifying the result by hand, most importantly removing the flowers and the curtain from the image[2]. This approach suffers from the

[2]it is difficult to tell whether the bottom right hand corner of the image is part of the curtain or the cardigan. In the hint image for these results it has been marked as foreground

disadvantage that estimating alpha for images with a bluescreen background is only one special case of alpha estimation, and does not represent performance on other images.

While neither of these solutions are ideal as quantitive measures, both have been adopted for the quantitive results presented here.

6.6.2 Bluescreen Results

Results produced by each of the six algorithms from the bluescreen *Rachael* image are shown in Fig. 6.5 to Fig. 6.12. Parameters used for the Principal Axis algorithm are shown in Table 6.1. Similar parameters were used for other algorithms where applicable. The alpha channel is shown alongside a composition of the alpha channel and clean foreground image over an artificial background. Areas where certain algorithms perform poorly are enlarged to show detail.

Fig. 6.5(b) is a processed version of the alpha channel produced by the bluescreen algorithm in order to highlight the algorithm's susceptibility to film grain noise in the background area.

Table 6.2 compares the Root Mean Squared Error (RMSE) difference between the alpha channel produced by each algorithm and the hand–generated Ground Truth alpha channel. Since all pixels marked as clean background or foreground are automatically marked as such in the alpha channel, only the unknown pixels are considered in the RMSE calculation. Whether the flowers are considered background or foreground is therefore ignored in the RMSE calculation, and only part of the curtain is considered. Three figures are given for each image: one for the errors in the alpha channel corresponding to the part of the foreground that covers the blue screen, one for the part that covers the curtain, and one for the whole image. This allows separate analysis of performance in the difficult curtain area with the blue screen area.

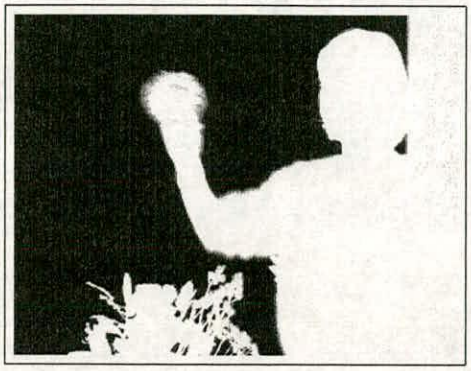
RMSE results are calculated from the 8 bit alpha channels directly. The greatest possible RMSE is therefore 255, and an algorithm producing random alpha values for each pixel would give an expected RMSE of 127.5.

As expected, the bluescreen alpha channel shows the best performance of all in the bluescreen area. Although the alpha channel is very noisy in the background area, the incorrect alpha values are close enough to the correct value to cause only a small error. The bluescreen algorithm shows very poor performance in the foreground area. The general algorithm with the best performance in the bluescreen area is the Principal Axis algorithm running with alpha

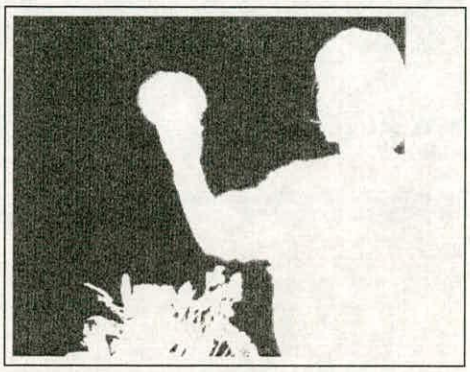
Parameter	Description	Value
blockwidth	size of block for averaging when growing to expand known areas (see section 5.12.1	10 pixels
growthresh	Threshold for region growing to expand known areas (see section 5.12.1	1% maximum intensity
backpoints	Number of pixels from nearby background area and previously estimated pixels used to form background cluster	300 pixels
forepoints	Number of pixels from nearby foreground area and previously estimated pixels used to form foreground cluster	300 pixels
minalphfore	Minimum alpha value required to use a previously estimated pixel in a foreground cluster	240/255
maxalphback	Maximum alpha value required to use a previously estimated pixel in a background cluster	15/255
Stripe width	Width of area from which background and foreground pixels can be selected to form clusters (only known pixels within this distance of the original unknown area will be used)	10 pixels

Table 6.1: *Parameters used for Principal Axis algorithm on the* Rachael *test image*

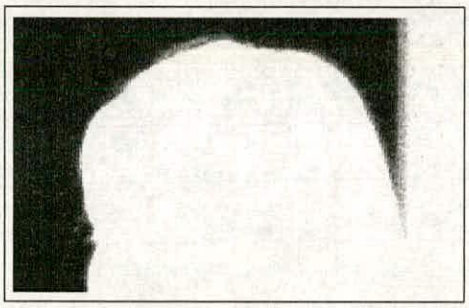
range adjustment (see section 5.14) although this algorithm performs poorly in the curtain area. Alpha range adjustment produces poor results when the foreground and background clusters overlap. *Photoshop* performs particularly well in this area, but mainly because the hint area was much smaller here, and the ground truth reference image was generated from the Photoshop result for this part of the image. The Genetic Algorithm does not perform as well as expected in the bluescreen area. It is often the case that the fitness values obtained by using the result from the Principal Axis algorithm are better than those evolved using the Genetic Algorithm. This suggests that the fitness function (Equation 6.3) is difficult to optimise using a GA for this particular image.



(a) Alpha channel



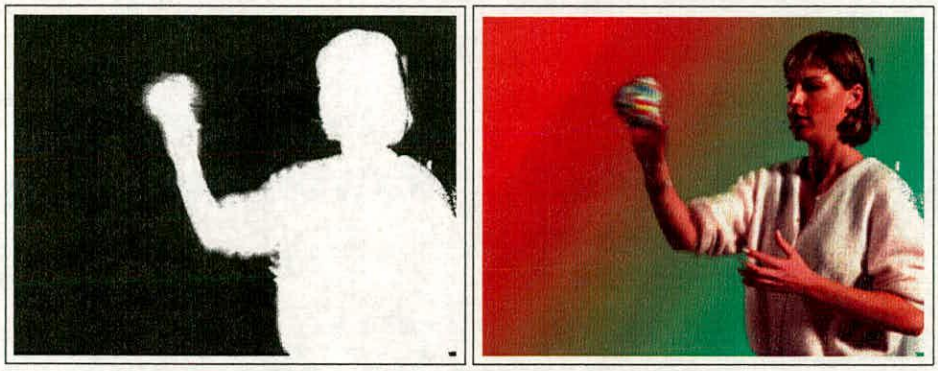
(b) Alpha channel brightened to show film grain noise



(c) Head area detail

(d) Hand area detail

Figure 6.5: *Results of running CFC's* Keylight *on the* Rachael *test image*

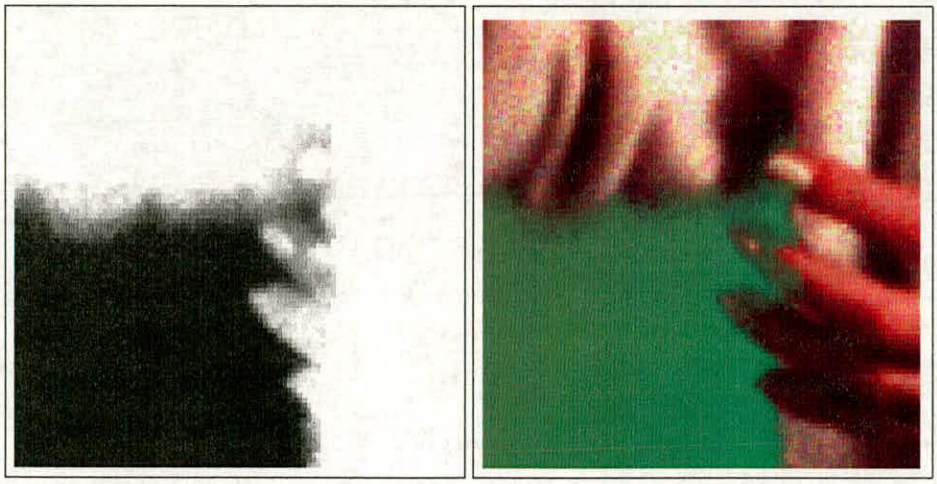


(a) Alpha channel

(b) Composite image

(c) Head area detail: Alpha Channel

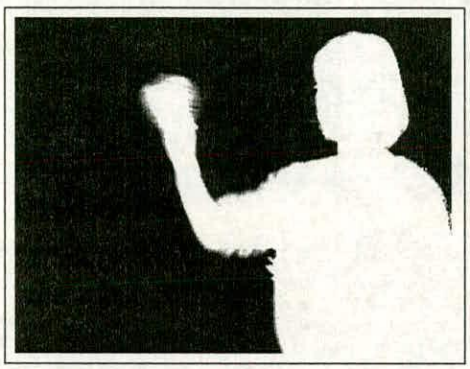
(d) Head area detail: Composite image



(e) Hand area detail: Alpha Channel

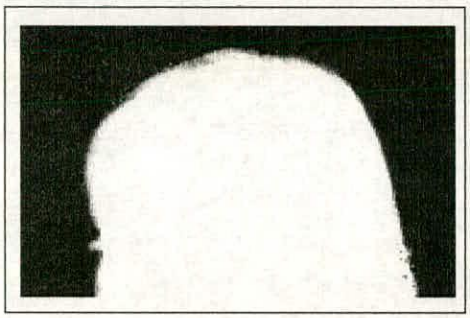
(f) Hand area detail: Composite image

Figure 6.6: *Results of running Corel* KnockOut *on the* Rachael *test image*



(a) Alpha channel

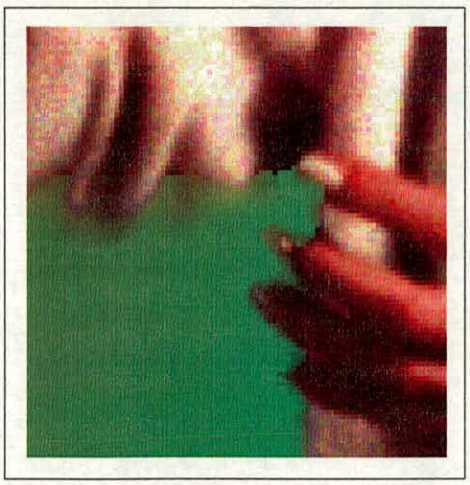
(b) Composite image



(c) Head area detail: Alpha Channel

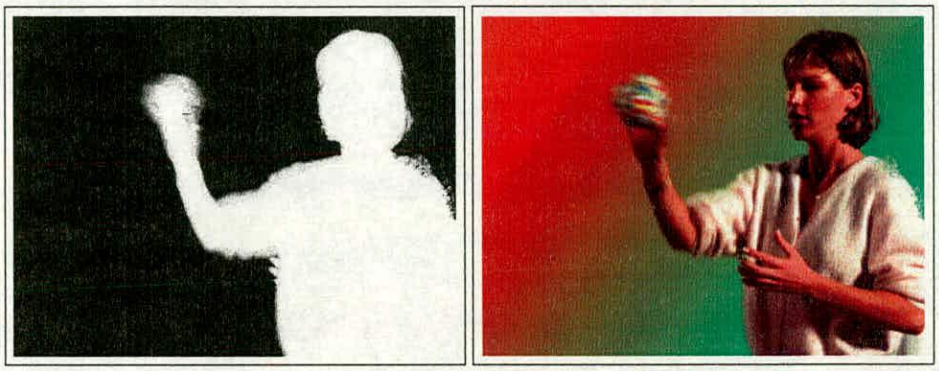
(d) Head area detail: Composite image

(e) Hand area detail: Alpha Channel

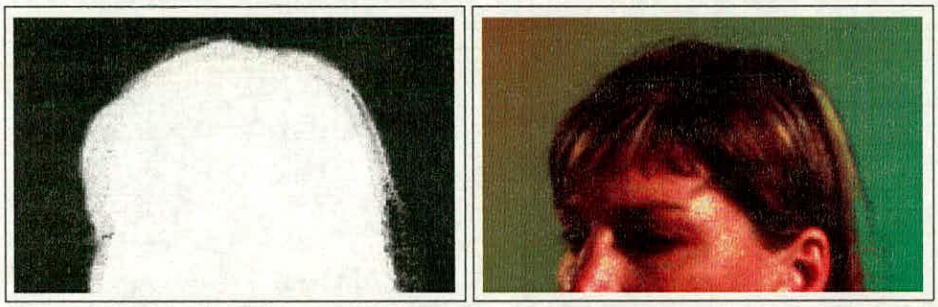


(f) Hand area detail: Composite image

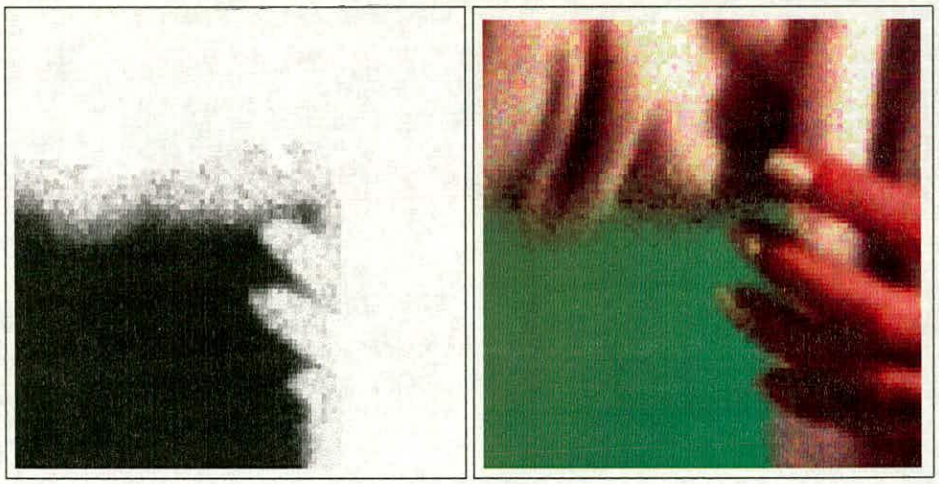
Figure 6.7: *Results of running Adobe* Photoshop *on the* Rachael *test image*



(a) Alpha channel **(b)** Composite image

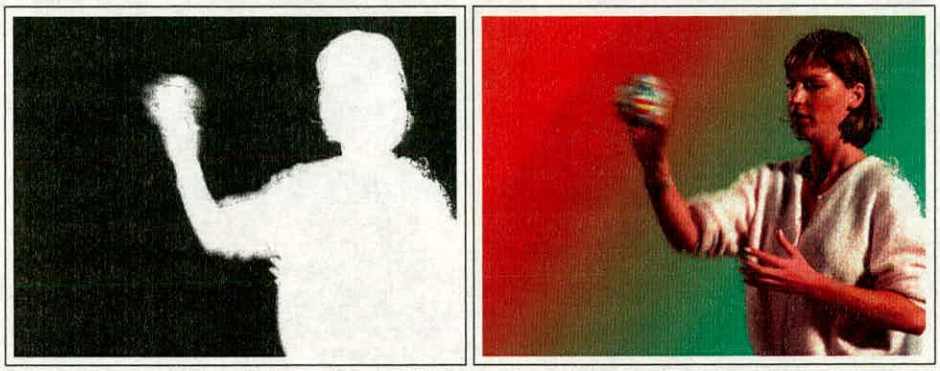


(c) Head area detail: Alpha Channel **(d)** Head area detail: Composite image



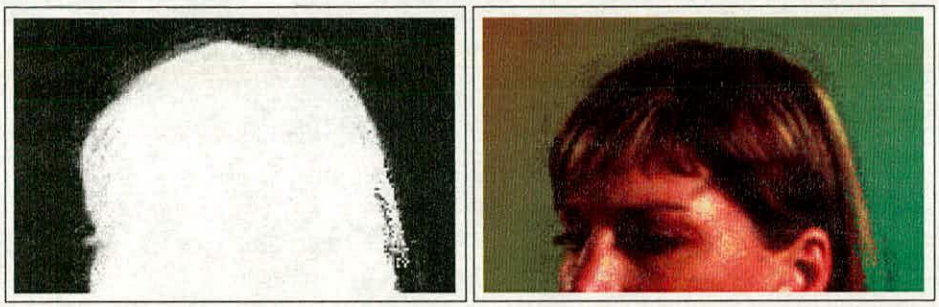
(e) Hand area detail: Alpha Channel **(f)** Hand area detail: Composite image

Figure 6.8: *Results of running Ruzon and Tomasi's algorithm on the* Rachael *test image*



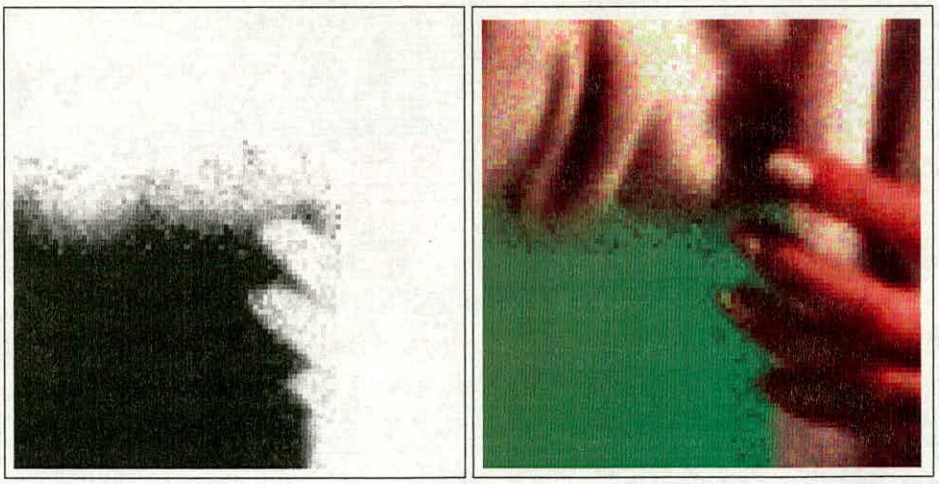
(a) Alpha channel

(b) Composite image



(c) Head area detail: Alpha Channel

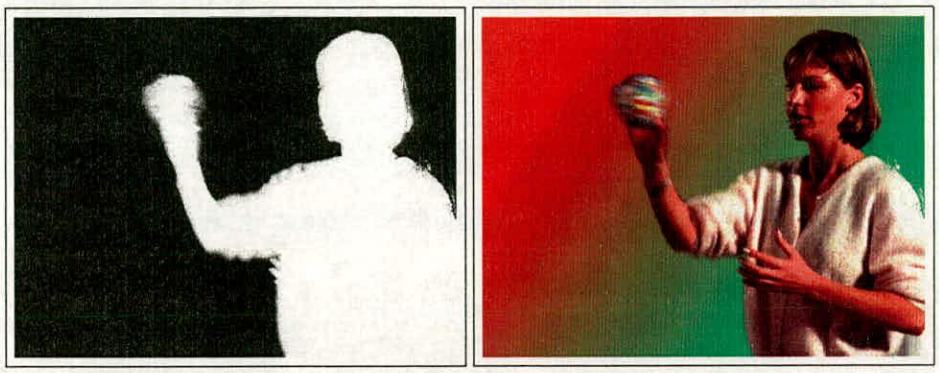
(d) Head area detail: Composite image



(e) Hand area detail: Alpha Channel

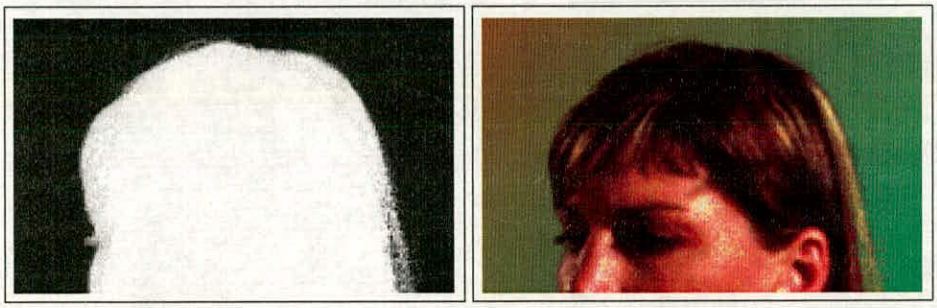
(f) Hand area detail: Composite image

Figure 6.9: *Results of running the algorithm of Chuang* et al *on the* Rachael *test image*



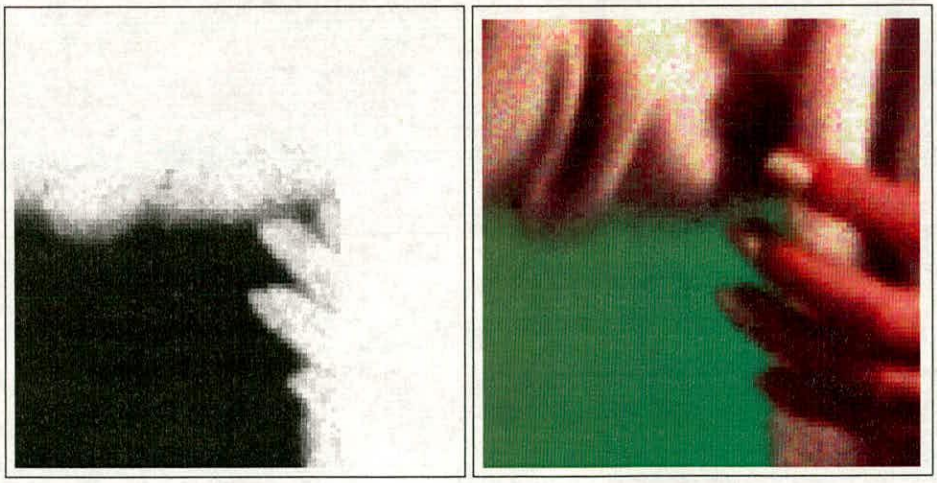
(a) Alpha channel

(b) Composite image



(c) Head area detail: Alpha Channel

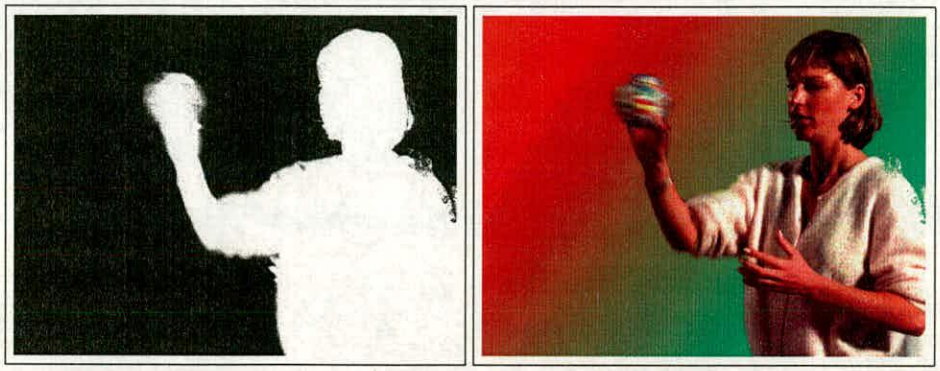
(d) Head area detail: Composite image



(e) Hand area detail: Alpha Channel

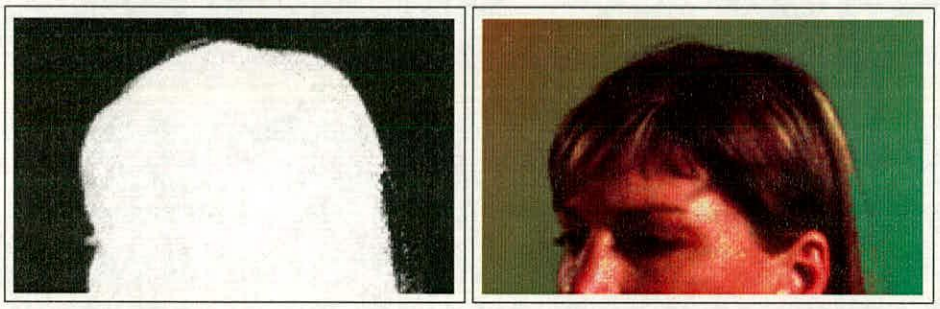
(f) Hand area detail: Composite image

Figure 6.10: *Results of running the Principal Axis algorithm without alpha correction on the* Rachael *test image*



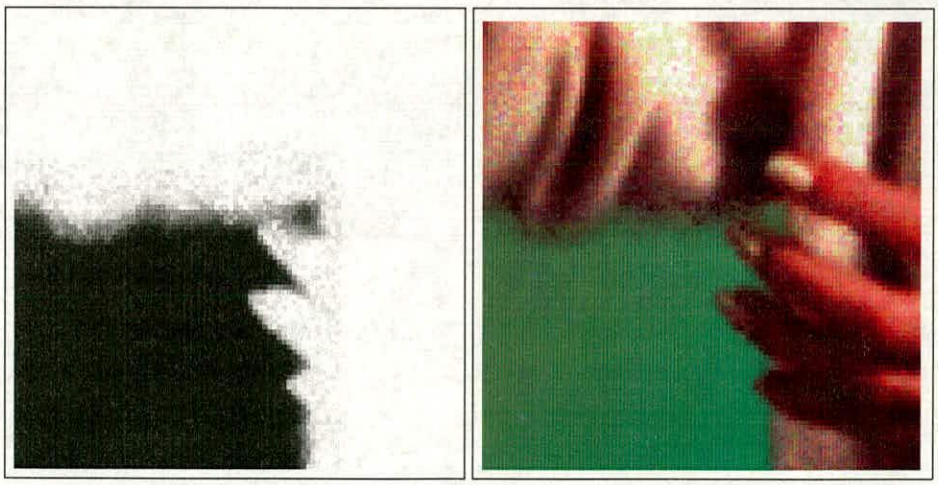
(a) Alpha channel

(b) Composite image



(c) Head area detail: Alpha Channel

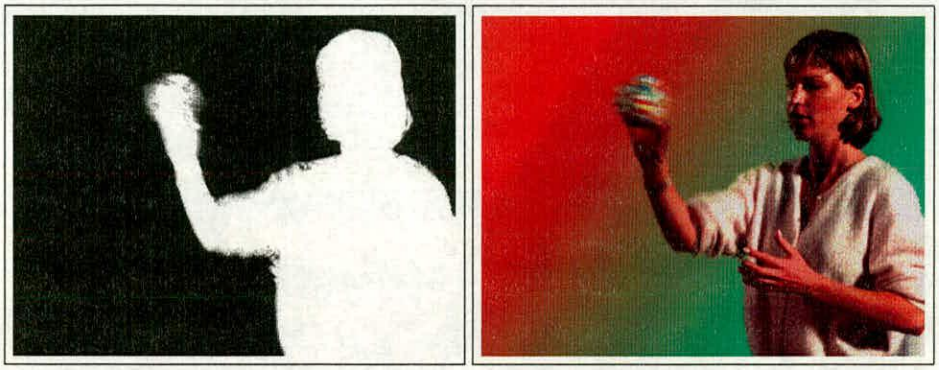
(d) Head area detail: Composite image



(e) Hand area detail: Alpha Channel

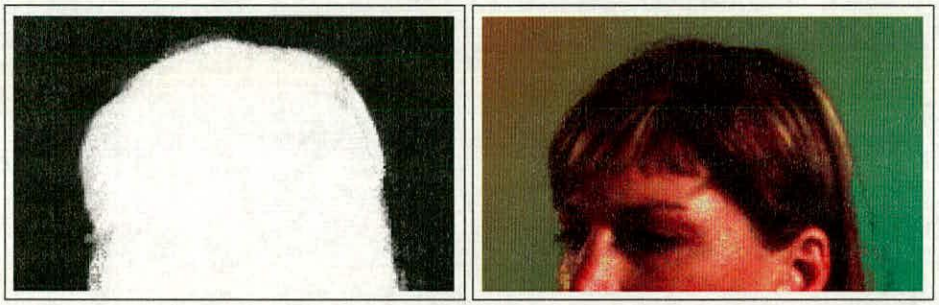
(f) Hand area detail: Composite image

Figure 6.11: *Results of running the Principal Axis algorithm with alpha correction on the* Rachael *test image*



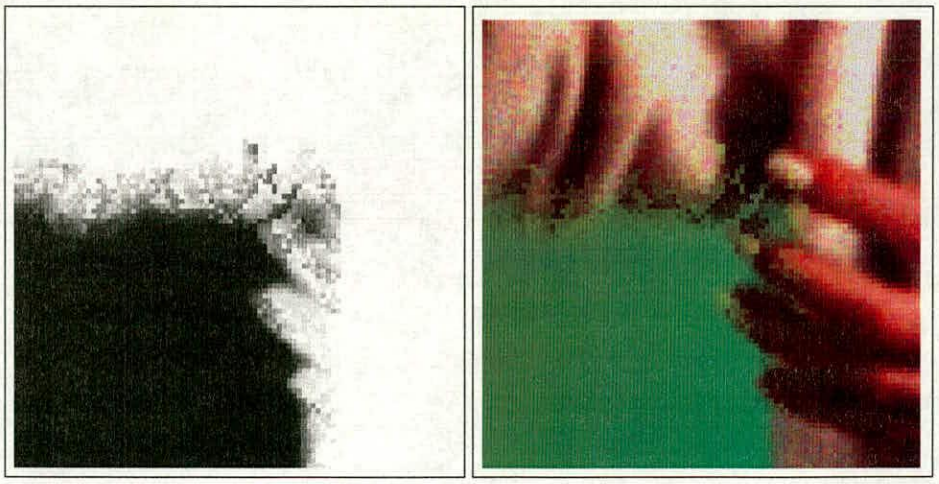
(a) Alpha channel

(b) Composite image



(c) Head area detail: Alpha Channel

(d) Head area detail: Composite image



(e) Hand area detail: Alpha Channel

(f) Hand area detail: Composite image

Figure 6.12: *Results of running the Genetic Algorithm on the* Rachael *test image*

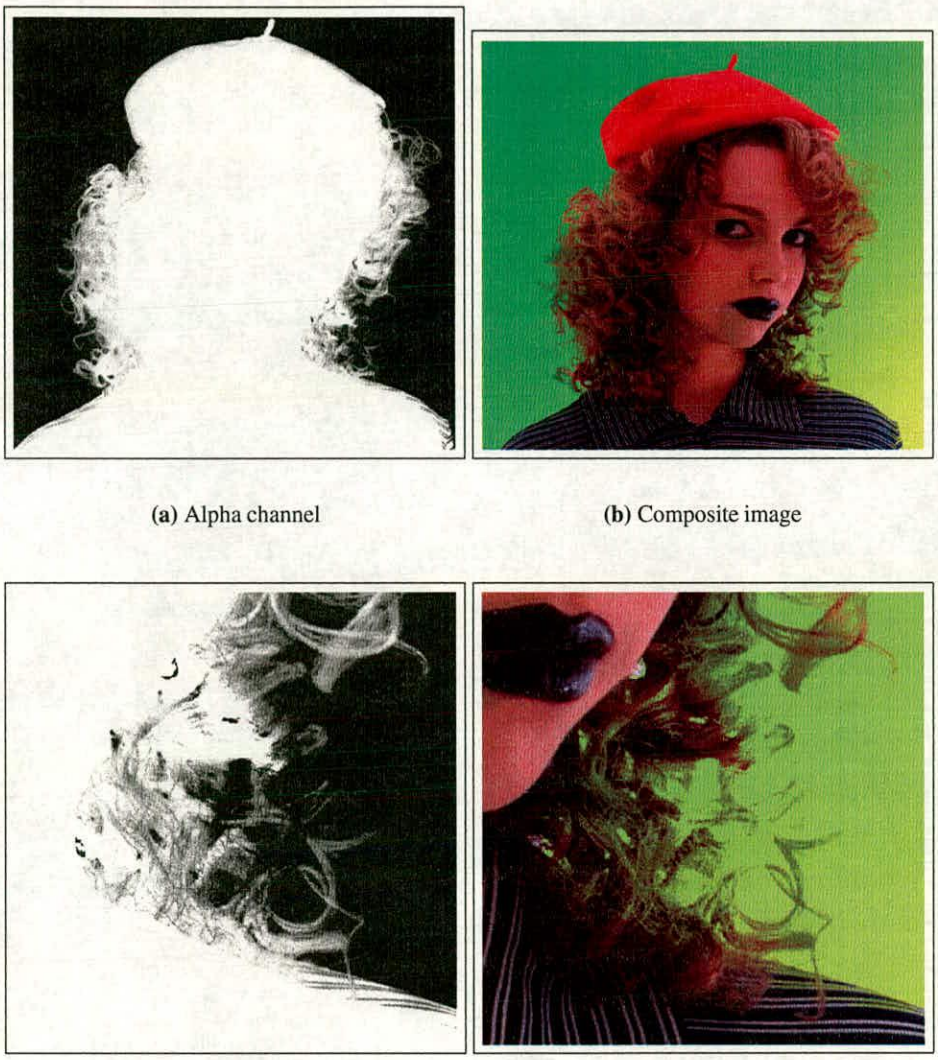
	RMSE		
Algorithm	**Bluescreen area**	**Curtain area**	**Whole image**
Bluescreen	10.45	124.8	62.83
KnockOut	47.96	77.3	56.68
Photoshop	50.88	40.26	48.46
Ruzon and Tomasi	40.30	72.31	50.19
Chuang *et al*	46.16	73.50	54.24
Principal Axis	37.69	54.22	42.39
Principal Axis (alpha adjustment)	35.53	80.66	50.62
Genetic Algorithm	50.85	58.51	52.85

Table 6.2: *RMSE performance of algorithms on the* Rachael *test image*

6.6.3 The *Edith* Image

Although a single colour white background would appear to be a trivial image to segment, the Edith image (fig. A.5) is challenging: The T-shirt has white stripes, which merge with the background. No algorithm was capable of segmenting these stripes correctly. Also, the hair has highlights, causing it to appear white in places.

The algorithm of Chuang *et al* — as it is detailed in their paper — cannot handle a background with zero variance. In this image, the background has neither detail nor noise (it is possible that the film and scanner were saturated, eliminating noise). The set of linear equations that are solved to find the foreground and background colours for any given alpha value becomes singular, and a foreground value cannot be found. Results of applying their algorithm to a modified version of the image are presented in Fig. 6.18(d).



(a) Alpha channel

(b) Composite image

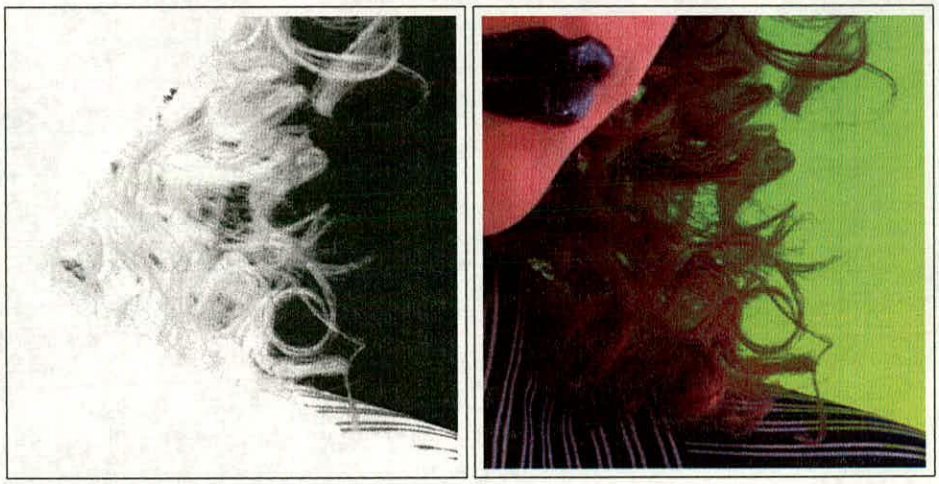
(c) Left shoulder area detail: alpha channel

(d) Left shoulder area detail: composite

Figure 6.13: *Results of running Adobe* Photoshop *on the* Edith *image*

(a) Alpha channel

(b) Composite image



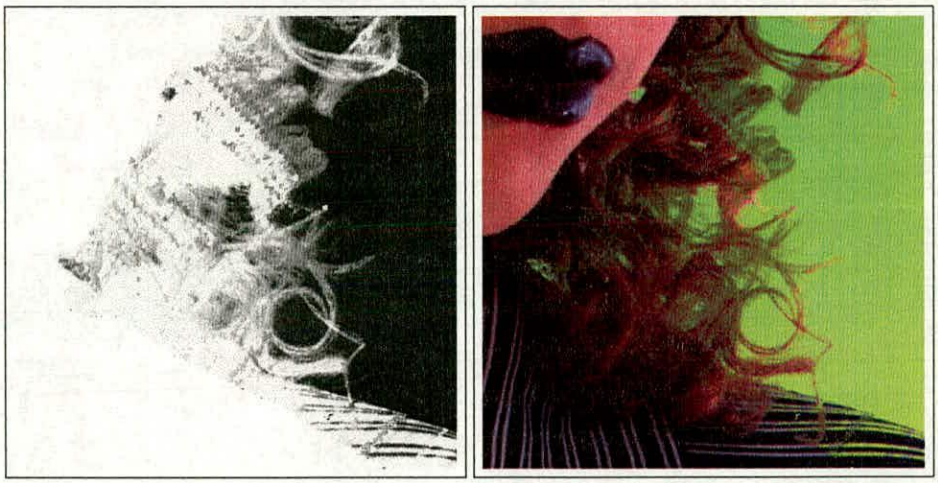
(c) Left shoulder area detail: alpha channel

(d) Left shoulder area detail: composite

Figure 6.14: *Results of running Corel* KnockOut *on the* Edith *image*

(a) Alpha channel

(b) Composite image



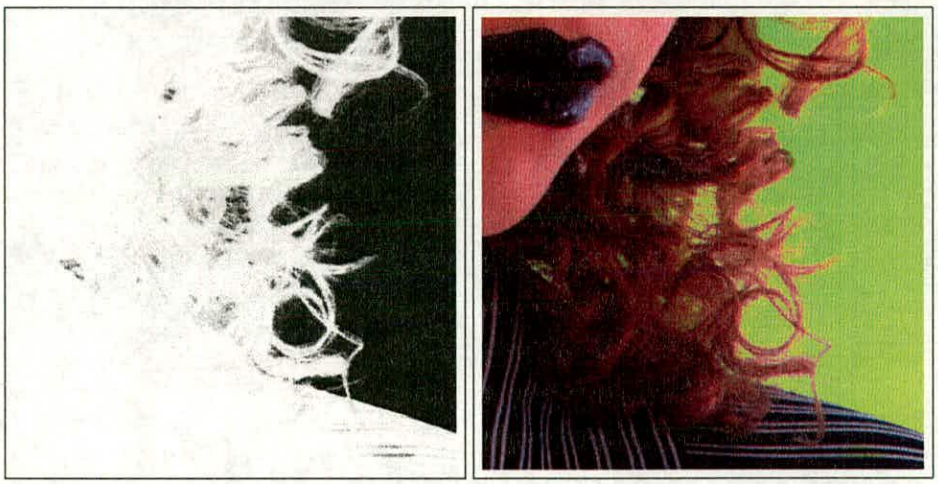
(c) Left shoulder area detail: alpha channel

(d) Left shoulder area detail: composite

Figure 6.15: *Results of running Ruzon and Tomasi's algorithm on the* Edith *image*

(a) Alpha channel

(b) Composite image



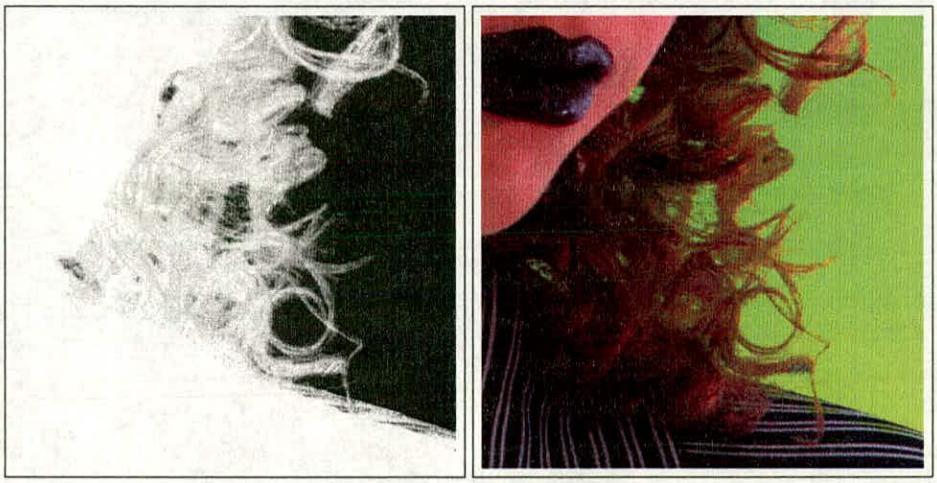
(c) Left shoulder area detail: alpha channel

(d) Left shoulder area detail: composite

Figure 6.16: *Results of running the Principal Axis algorithm on the* Edith *image*

(a) Alpha channel

(b) Composite image



(c) Left shoulder area detail: alpha channel

(d) Left shoulder area detail: composite

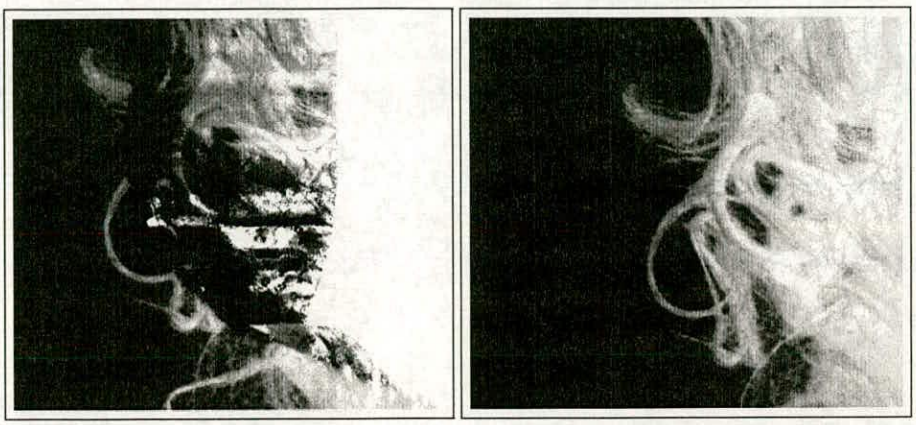
Figure 6.17: *Results of running Genetic algorithm on the* Edith *image*

6.6.3.1 Quantitive results for the *Edith* Image

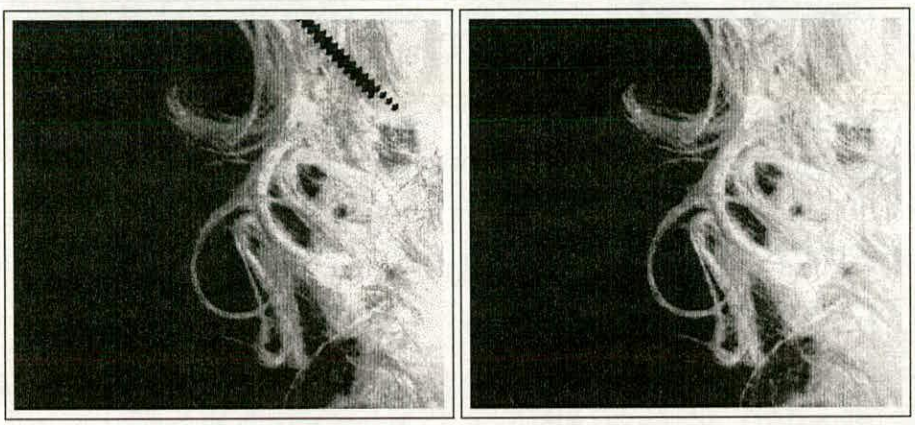
In order to obtain another numerical measure of performance, the composite image obtained from the Genetic Algorithm (Fig. 6.17(b)) was segmented using each algorithm. The result for this algorithm was used since it produced the result with very few artifacts. Since the alpha channel used to create this composite is known, the alpha channel produced by each algorithm can be compared to this result. Table 6.3 shows the Root Mean Squared Error for each algorithm. Fig. 6.18 shows details of the alpha channels produced. Since the reference alpha channel (shown in Fig. 6.18(f)) was created by the Genetic Algorithm, RMSE accuracy of the Genetic Algorithm is not comparable.

	RMSE		
Algorithm	**Top of image**	**T-shirt area**	**Whole image**
KnockOut	9.277	17.45	10.83
Photoshop	113.4	109.1	112.9
Ruzon and Tomasi	21.78	31.17	23.35
Chuang *et al*	9.077	15.01	10.14
Principal Axis	8.599	9.070	8.667
(Genetic Algorithm	5.898	7.057	6.078)

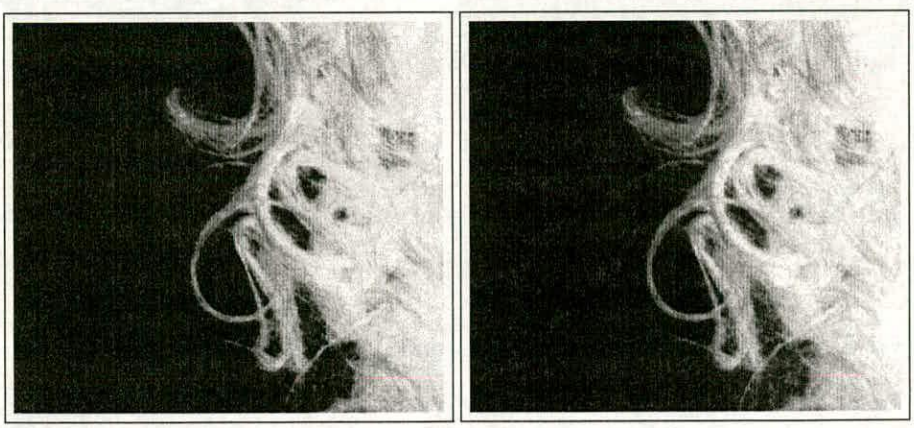
Table 6.3: *RMSE accuracy of alpha channels produced from composited* Edith *image*



(a) *Photoshop* **(b)** *KnockOut*



(c) Ruzon and Tomasi **(d)** Chuang *et al*



(e) Principal Axis **(f)** True alpha channel

Figure 6.18: *Details of alpha channels produced from composited* Edith *image*

6.6.4 The *Teddy* Image

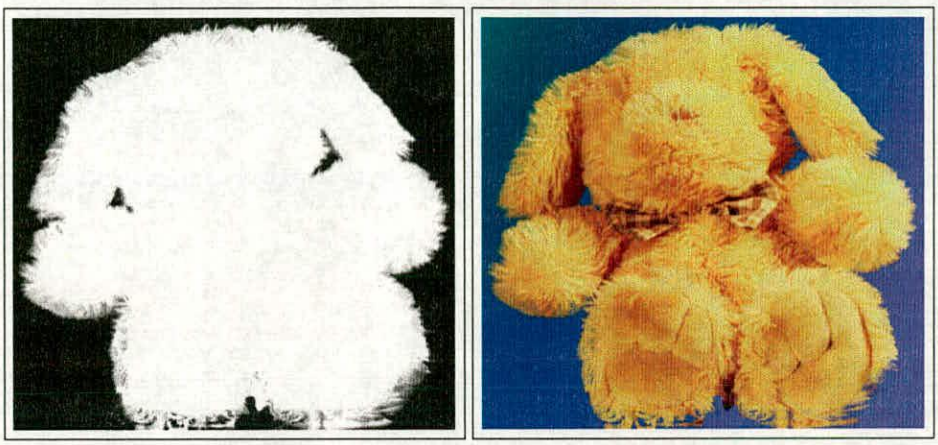
The performance of the algorithms with the *Teddy* image are shown in figs. 6.19 to 6.24. All results show good performance against the background (which is textured but relatively uniform and hence is rather similar to a bluescreen segmentation), but the colour of the table is very close to the colour of the fur, which causes all algorithms to perform poorly in places. All algorithms also perform poorly around the back paws, where reflection off the bear onto the paper reduces performance.

Table 6.4 shows the timing of these algorithms on the same computer. Results were obtaining measuring the time spent on user mode using the Linux `time` command. All algorithms could be heavily optimised, and the computer used is not particularly high performance, but the relative timings indicate the difference in computational complexity between the algorithms. The implementation of Ruzon and Tomasi in particular could probably be made significantly more efficient. Results for the KnockOut and Photoshop packages are not shown since these packages do not run under Linux. The input hint image for these programs was also different. Both programs processed the frame in about ten seconds. Since the other algorithms were all implemented on the same system re-using code between the programs were possible, comparison between these programs is fair.

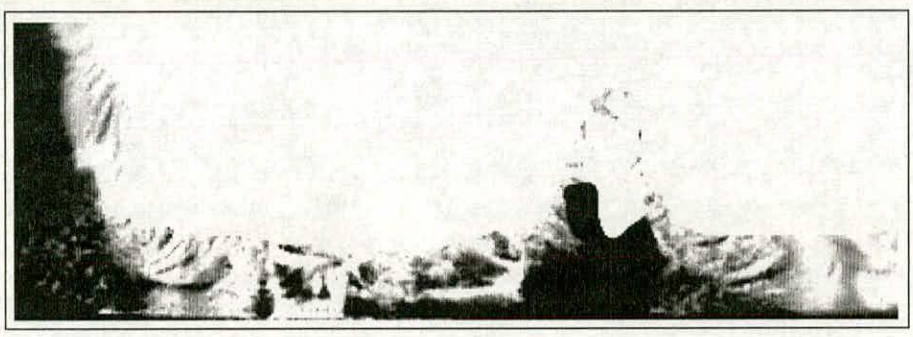
Algorithm	Run time
Ruzon and Tomasi	2h45m
Chuang *et al*	32m
Principal Axis	6m
Genetic Algorithm	45m

Table 6.4: *Run times of unoptimised implementations of algorithms on* Teddy *image*

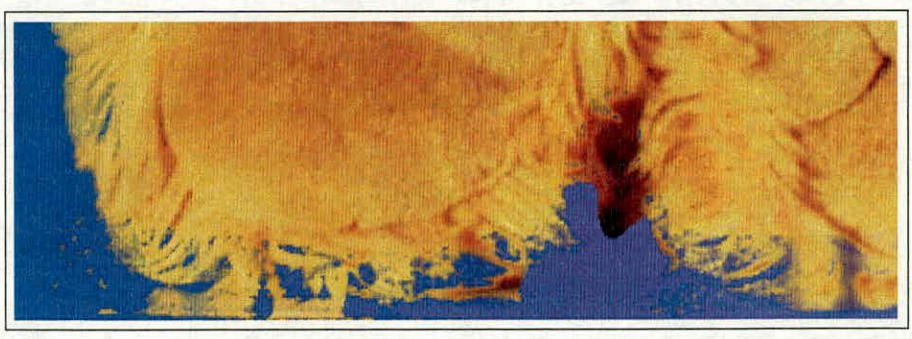
Clearly, the Principal Axis algorithm is significantly faster than the alternatives implemented under Linux, and arguably produces one of the best results.



(a) Alpha channel **(b)** Composite image

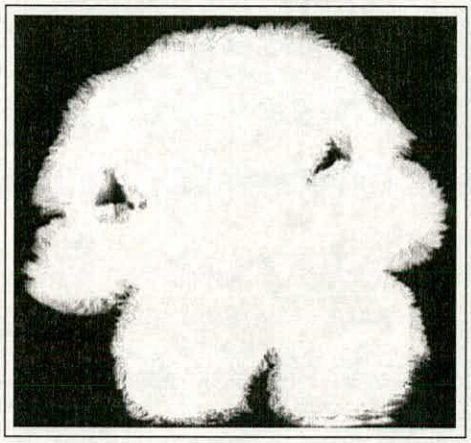


(c) Back paw area detail: alpha channel

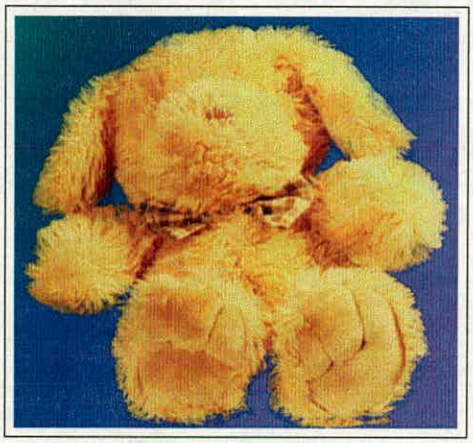


(d) Back paw area detail: composite image

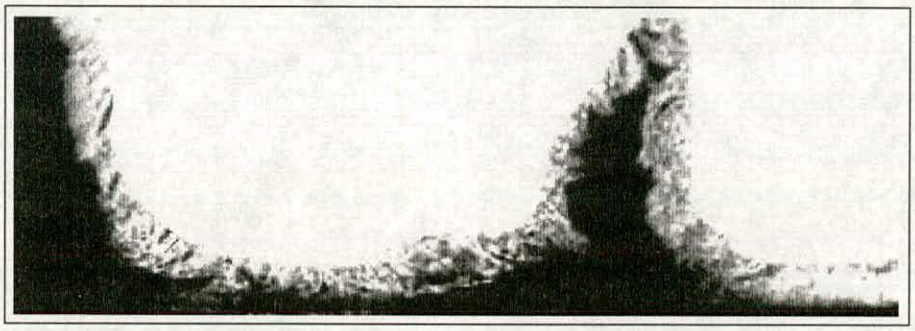
Figure 6.19: *Results of running Adobe* Photoshop *on the* Teddy *image*



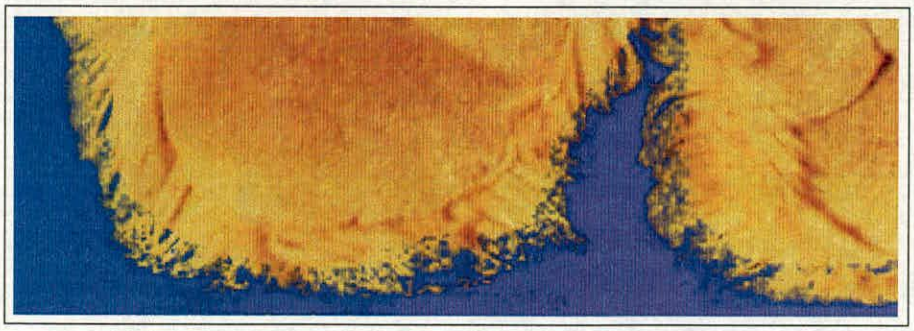
(a) Alpha channel



(b) Composite image

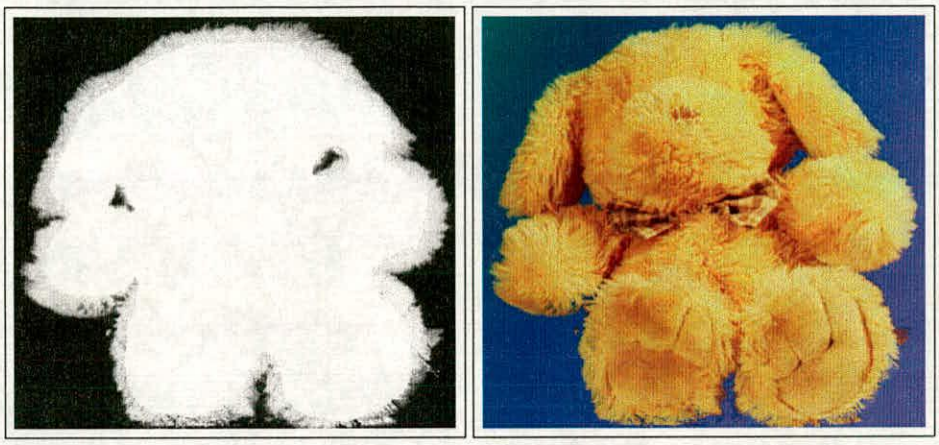


(c) Back paw area detail: alpha channel



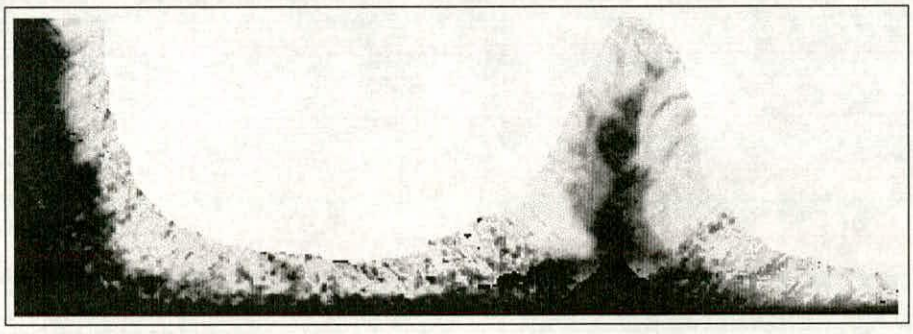
(d) Back paw area detail: composite image

Figure 6.20: *Results of running Corel* KnockOut *on the* Teddy *image*

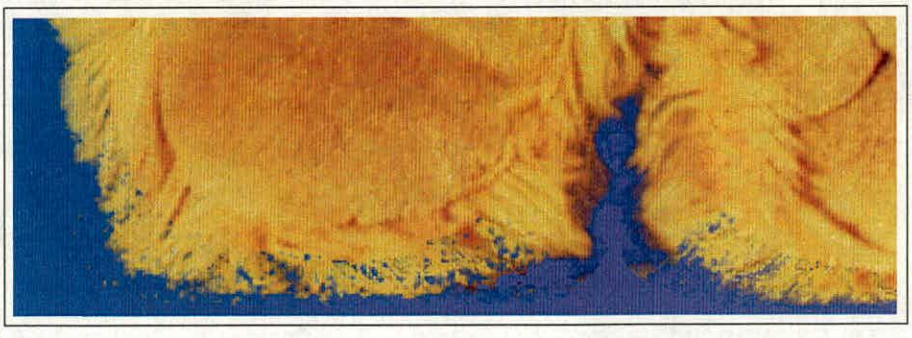


(a) Alpha channel

(b) Composite image

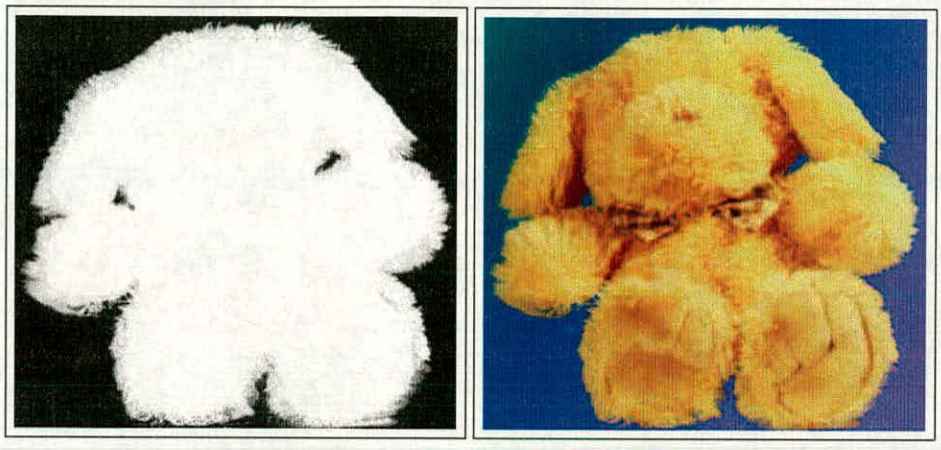


(c) Back paw area detail: alpha channel



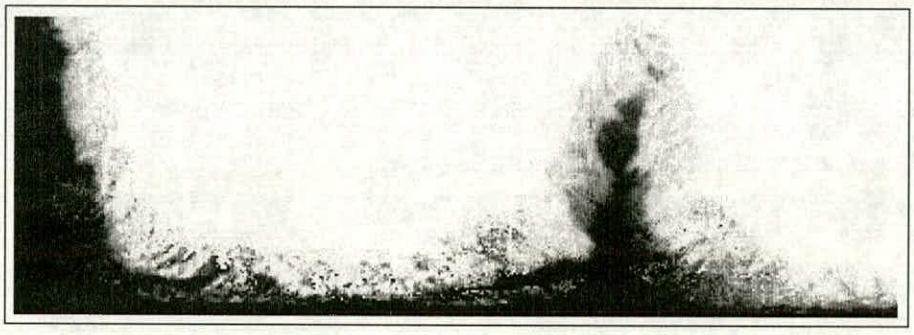
(d) Back paw area detail: composite image

Figure 6.21: *Results of running Ruzon and Tomasi's algorithm on the* Teddy *image*

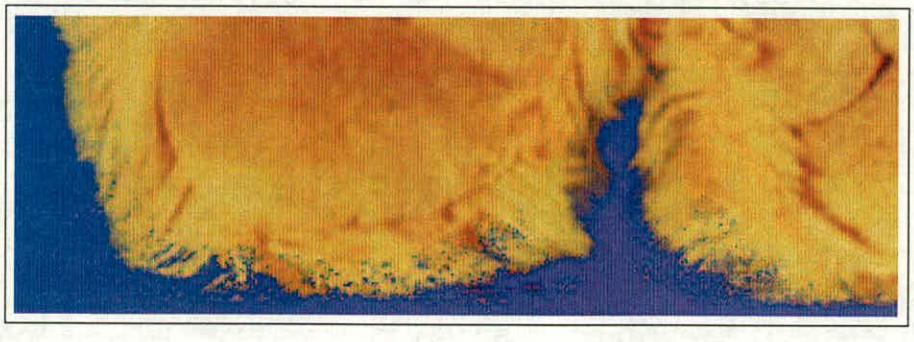


(a) Alpha channel

(b) Composite image

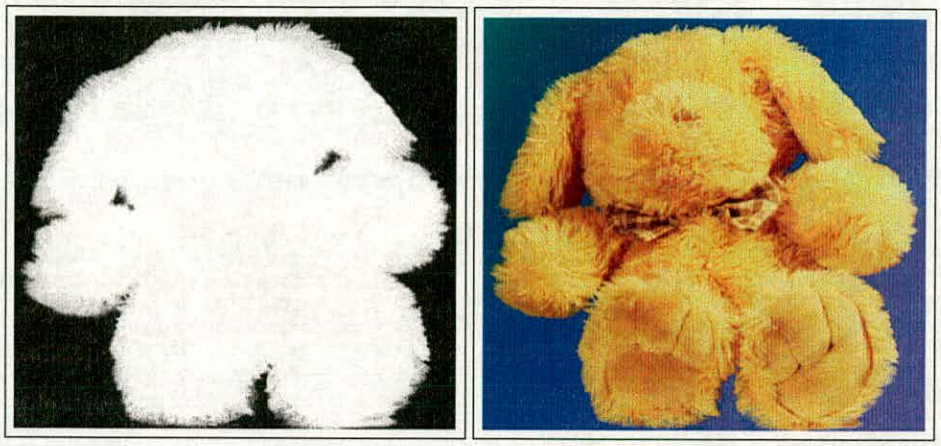


(c) Back paw area detail: alpha channel



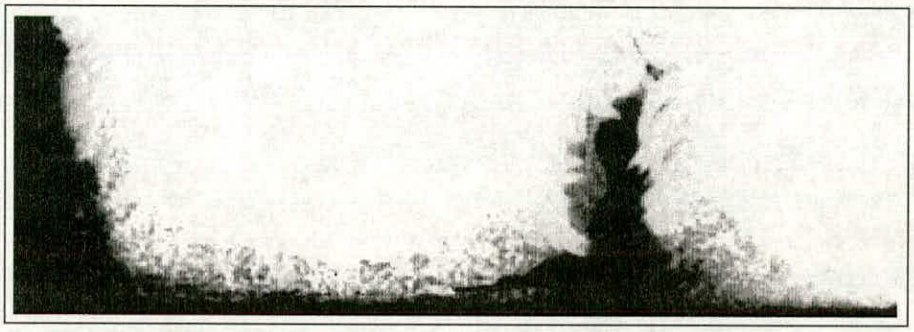
(d) Back paw area detail: composite image

Figure 6.22: *Results of running the algorithm of Chuang* et al *on the* Teddy *image*

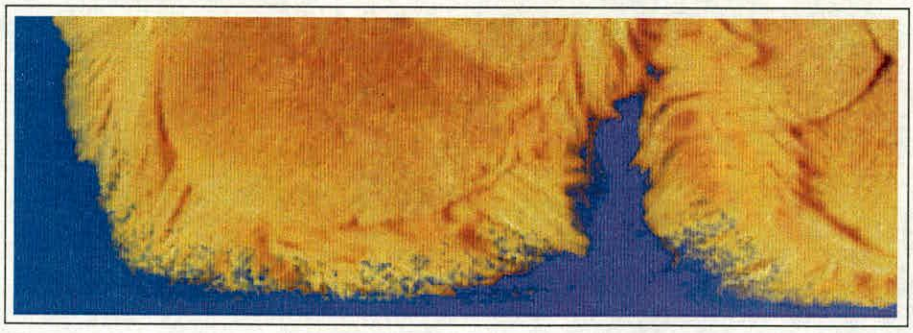


(a) Alpha channel

(b) Composite image

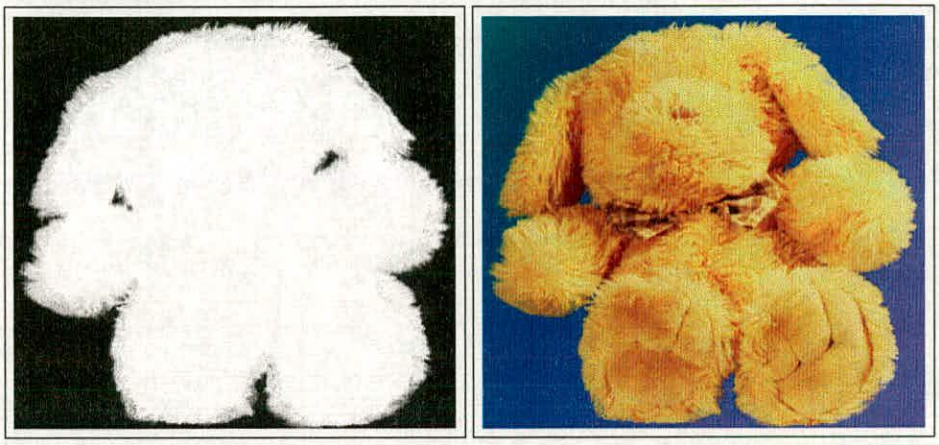


(c) Back paw area detail: alpha channel

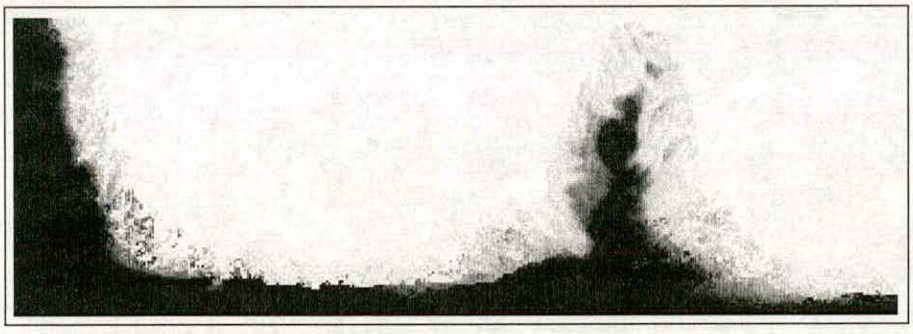


(d) Back paw area detail: composite image

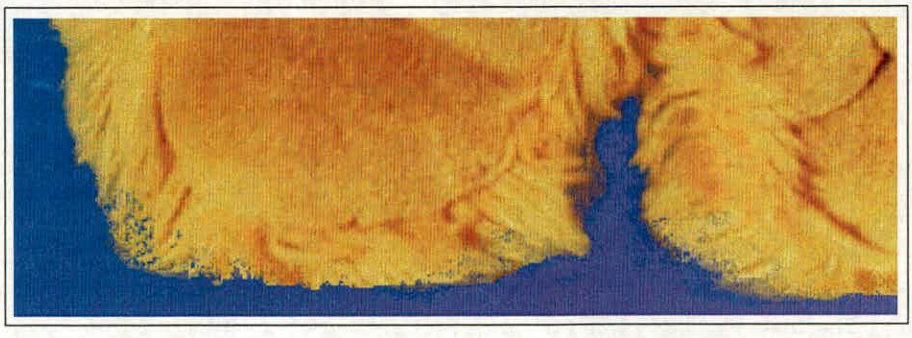
Figure 6.23: *Results of running the Principal Axis algorithm on the* Teddy *image*



(a) Alpha channel **(b)** Composite image



(c) Back paw area detail: alpha channel

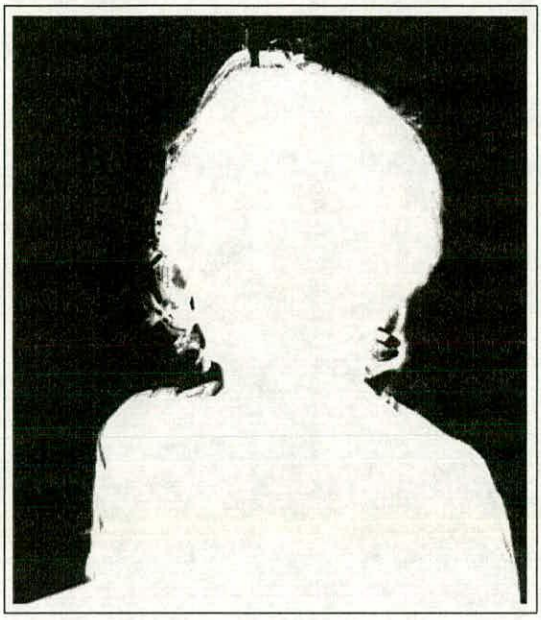


(d) Back paw area detail: composite image

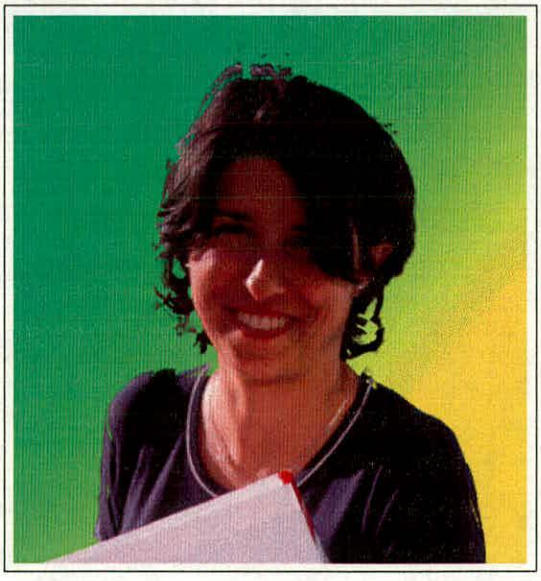
Figure 6.24: *Results of running Genetic algorithm on the* Teddy *image*

6.6.5 The *Gema* Image

Results for the Gema image are shown in figs. 6.25 to 6.30. The algorithms cope differently with the strands of hair around the top of the head, and by the right ear. The *Photoshop* and *KnockOut* algorithms both produce large artifacts in the alpha channel. The algorithms of Chuang *et al* and Ruzon and the Genetic Algorithm seem to produce alpha values that are a little too low. The results from the Principal Axis algorithm and Ruzon and Tomasi seem to be best.

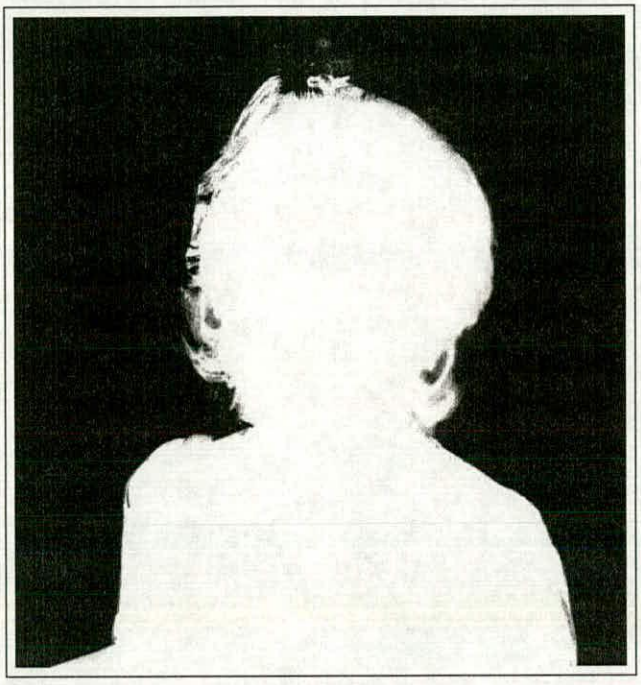


(a) Alpha channel

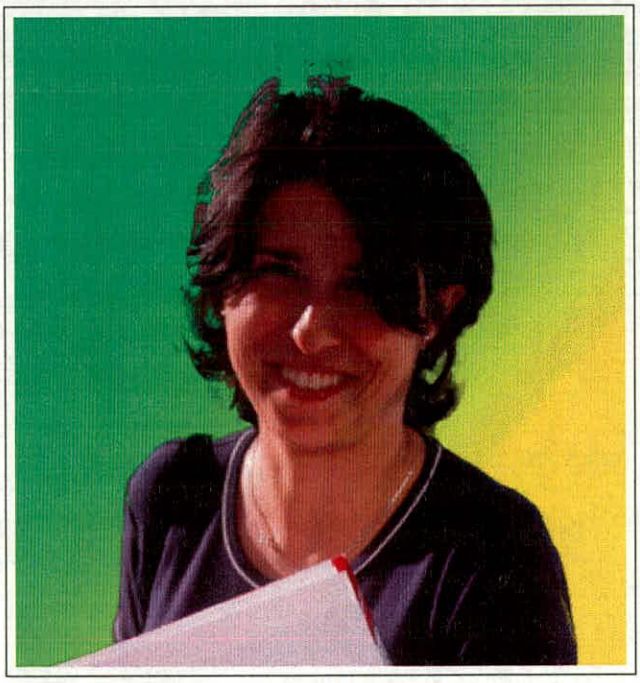


(b) Composite image

Figure 6.25: *Results of running Adobe* Photoshop *on the* Gema *image*

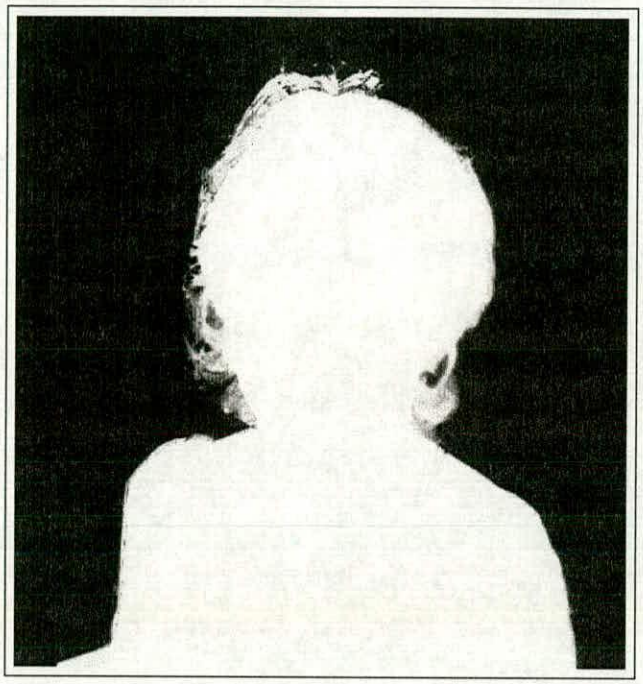


(a) Alpha channel

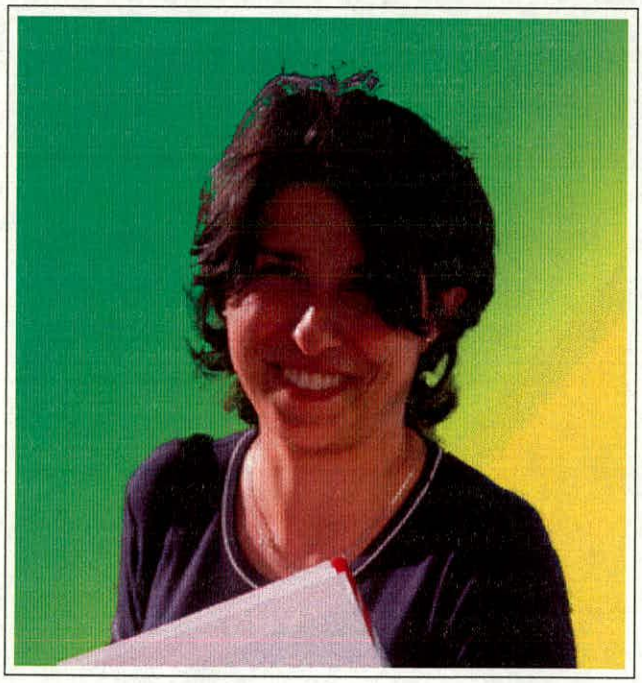


(b) Composite image

Figure 6.26: *Results of running Corel* KnockOut *on the* Gema *image*

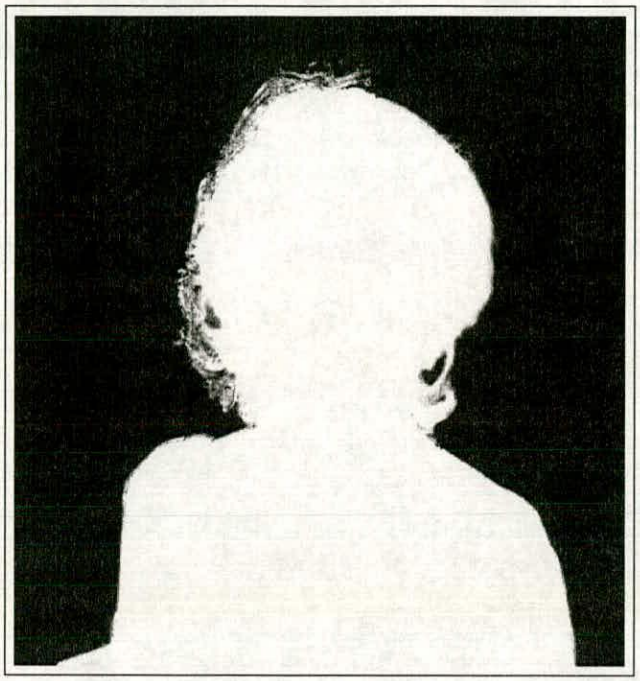


(a) Alpha channel

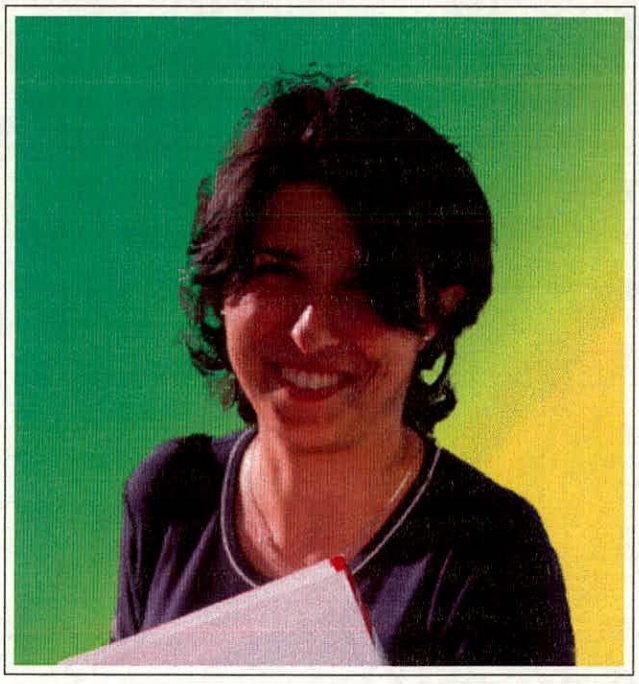


(b) Composite image

Figure 6.27: *Results of running Ruzon and Tomasi's algorithm on the* Gema *image*

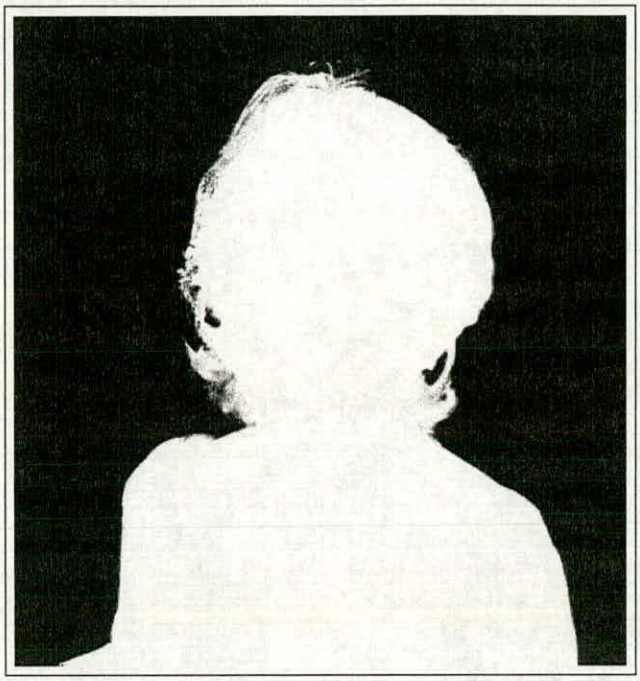


(a) Alpha channel

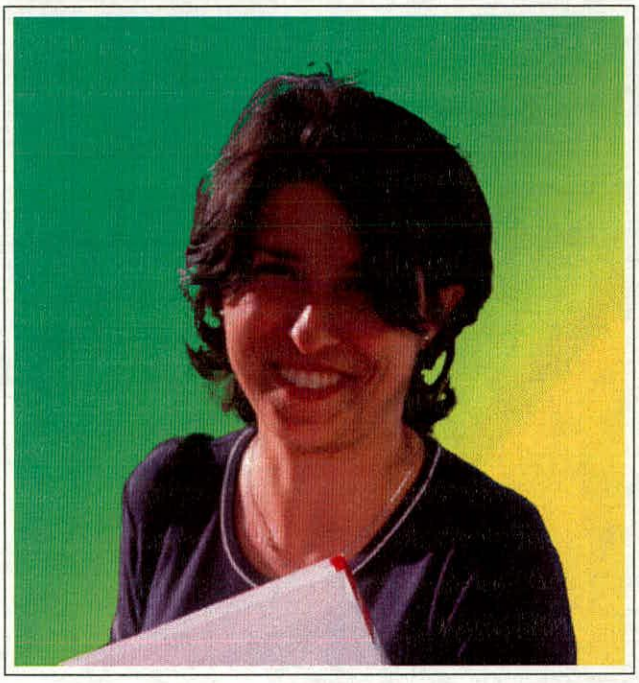


(b) Composite image

Figure 6.28: *Results of running the algorithm of Chuang* et al *on the* Gema *image*

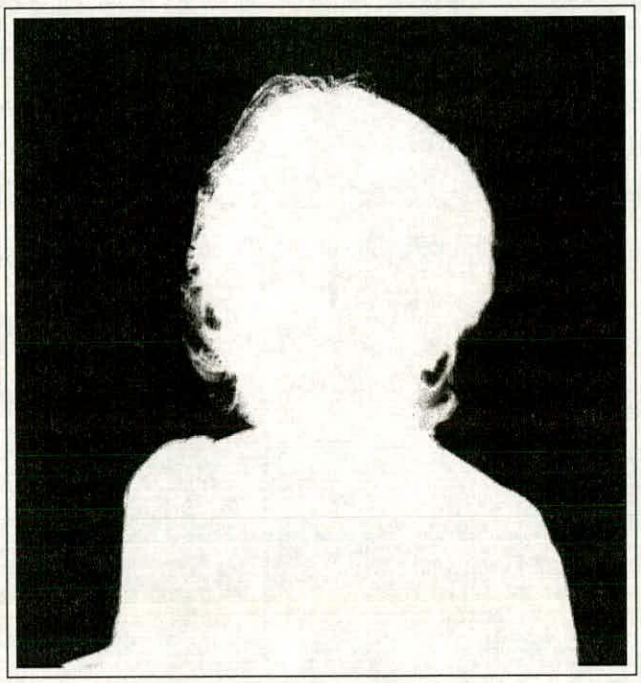


(a) Alpha channel

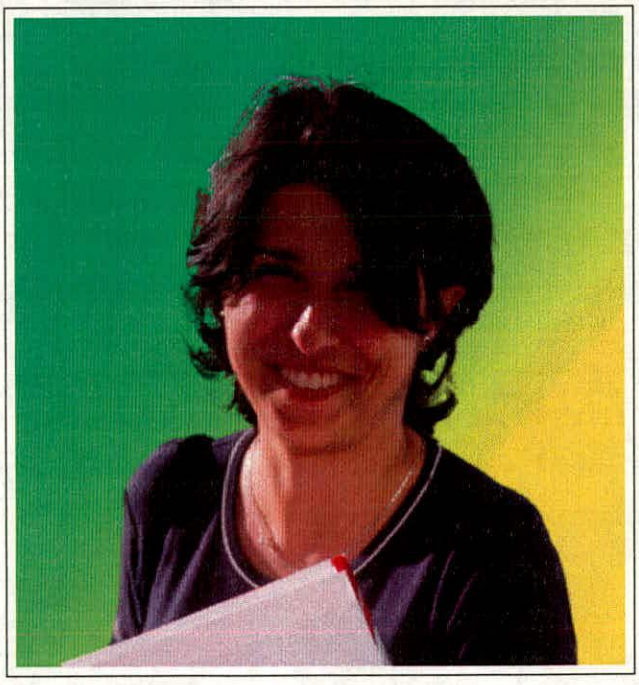


(b) Composite image

Figure 6.29: *Results of running the Principal Axis algorithm on the* Gema *image*



(a) Alpha channel



(b) Composite image

Figure 6.30: *Results of running Genetic algorithm on the* Gema *image*

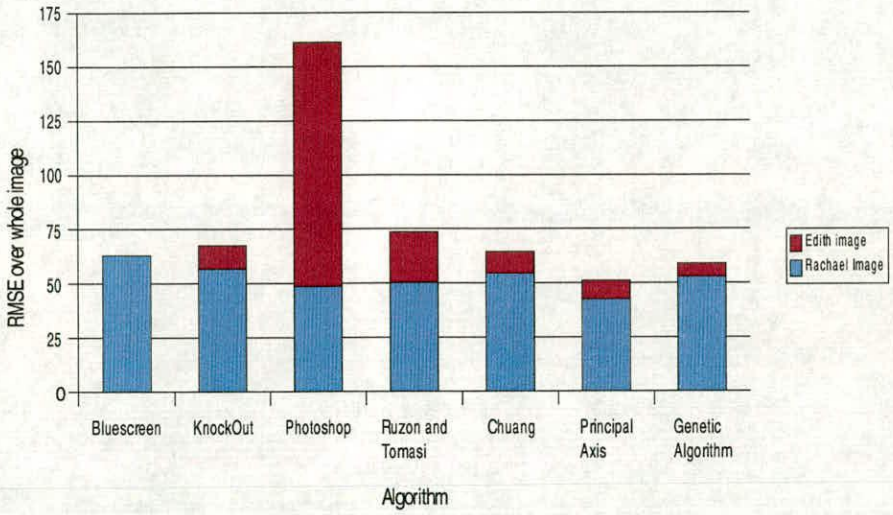


Figure 6.31: *Chart of RMSE error of each algorithm combined for both test images. The bluescreen algorithm was not run on the Edith image.*

6.7 Summary

This chapter has introduced three existing techniques for alpha channel estimation, two of which use Gaussian Mixture Models and a Bayesian framework, but apply simplifications in order to solve the problem rapidly. A more thorough Genetic Algorithm based solution to the Bayesian Gaussian Mixture Model matting problem has been proposed. A novel RMSE–based technique was also developed in order to quantify performance of alpha channel estimation.

Results from testing all four of these algorithms and the two commercial packages that perform alpha channel estimation reveal no clear winner. While the method of testing is not ideal, quantitive results suggest that the Principal Axis algorithm performs slightly better than any of the others, and the Genetic Algorithm also performs very well. However, ground truth images were not easy to generate and the result produced by the RMSE algorithm often does not reflect certain artifacts in the result (for example the noise in the background area in fig. 6.5(b)).

Where no quantitive results are available, visual inspection reveals that the Principal Axis and Genetic algorithms generally produce results with fewer artifacts. The other algorithms described in this chapter often produce results where the edges of the unknown area in the hint image are clearly visible, where there are geometric artifacts such as streaking or large parts

of foreground missing. Neither KnockOut nor Photoshop were able to resolve the individual strands of hair at the top of the *Gema* image. Similar failures of the algorithms in Chapter 5 caused them to be discarded in favour of the Principal Axis algorithm.

Chapter 7
Adaptation of Alpha Estimation to Image Sequences

7.1 Introduction

The algorithms presented in the previous chapters have been used to segment individual images. However, for motion picture special effects it is necessary to segment moving images (image sequences). It would be possible to segment an entire moving image sequence by requiring a human operator to generate a hint image for every separate frame of the sequence and segmenting them as independent still frames. Even if the tool for generating the hint image is efficient, it would be preferable to eliminate (or at least significantly reduce) the amount of user input required to segment the sequence. This is the motivation for developing techniques for Image Sequence Alpha Estimation.

In this chapter, four such techniques for automatic hint image generation are presented. The chapter is structured as follows. Section 7.2 discusses two different approaches to alpha channel estimation in moving images, and concludes that automatic update of hint images is the simplest and most reliable approach. Estimating the hint image is similar to standard binary foreground/background segmentation, except that an unknown area must also be produced. Section 7.3 discuses different approaches to handling the unknown area in a hint image. Section 7.4 examines the possibility of using standard optical flow code to perform hint image update. Section 7.5 examines some of the problems with the application of optical flow estimation in motion picture resolution sequences.

Using motion estimation to perform image segmentation is a standard technique, similar to the work of Mitsunaga *et al* [97]. The drawbacks of such techniques when applied to motion picture resolution images suggest the development of a novel approach to hint image estimation. Section 7.6 considers the processing of local colour distributions to update the hint image. Section 7.6.1 presents a very simple algorithm to perform this operation. Since this algorithm is very slow, a system to speed it up using a novel *bounding set* representation of colour dis-

tributions is described in Section 7.6.2. Section 7.6.3 describes an alternative representation that produces much faster results at the expense of significant errors. Section 7.6.4 uses Gaussian Mixture Models to generate hint images in a Maximum Likelihood fashion. Since all of these techniques produce noisy results, a noise removal step is required. Section 7.7 describes a technique for removing noise without degrading the hint image.

It is unlikely that a whole sequence can be segmented successfully using a single frame. Section 7.9 considers the use of more than one reference frame in a sequence. Section 7.10 shows a graphical overview of a system that can segment motion picture sequences. Results of using such a system are presented in Section 7.11. The chapter is summarised in Section 7.12.

7.2 Hint Image Estimation vs. Alpha Channel Estimation

Two differing approaches for adapting alpha channel estimation to image sequences are

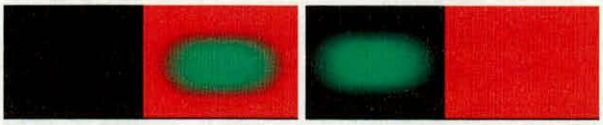
Direct approach: generate the alpha channel and clean foreground image for each frame directly, using one or more previously segmented neighbouring frames.

Hint image approach: generate a hint image for each frame using one or more previously segmented neighbouring frames

The former technique is much more complex than the latter. In single frame alpha estimation, it is possible to extract a small number of plausible candidates for the clean background and foreground colours, because there will be areas nearby that are known to be entirely background or foreground. With a moving image, there are no known clean foreground and background pixels. It is possible to select clean background and foreground pixels from the previous frame and estimate alpha from these, but this tends to be extremely computationally intensive and have poor accuracy.

Further, noise is more difficult to remove from an alpha channel and a clean foreground image than it is from a hint image, since a hint image can be processed using standard morphological operators that work on binary images.

Since the direct approach is likely to be slow, inaccurate and difficult to post-process to remove any errors, this chapter investigates the hint image approach.



(a) frame n **(b)** frame $n+1$

Figure 7.1: *The unknown area is not always consistent between frames*

7.3 Processing the Unknown Area

The alpha channel estimated for frame n can be considered to be a *ternary* segmentation of the frame: All pixels that are black in the alpha channel are background, all those that are white are foreground, and all the other grey pixels are considered to be unknown.

Using this ternary segmentation of frame n, there are several possible ways of extending binary segmentation to ternary hint image generation for frame $n+1$, so that the result has an unknown area as well as foreground and background:

Consider the unknown area as a separate region of the image. If some part p_q of frame $n+1$ matches to a part p_r of frame n and p_r is from the unknown area, mark p_q as unknown.

Use an uncertainty measure. Do not compare p_q to any part of the image that is marked as unknown. Instead, if the best match to any other part of frame n is uncertain, mark the p_q as unknown in the area.

Use morphological erosion. Again, do not compare to unknown parts of frame n and produce a binary segmentation. Form an unknown area by eroding away the background and foreground, replacing the eroded area with pixels marked as unknown.

The first of these appears simplest and most robust, but this is not always the case. Consider the case of Fig. 7.1, depicting a green fuzzy blob moving from right to left over a background that is red on one side and black on the other. In Fig. 7.1(a) the blob is over the red background and the unknown area appears yellow at its centre. In Fig. 7.1(b) the blob is entirely over the black background and the unknown area is dark green. Therefore, pixels that are in the unknown area in frame $n+1$ will not match to the unknown pixels in frame n and the first approach to handling pixels in the unknown area will fail.

Figure 7.2: *Motion vector field between frames 3 and 2 of the* Dragonheart *sequence*

The second case is particularly useful for disocclusions: There may be parts of frame $n+1$ that are not visible in frame n, for example the background revealed by a moving foreground. There should be no match for these parts in frame n, so motion data is incomplete. Marking these areas as unknown, and using the alpha estimation to generate an alpha value for them may solve this problem.

It is possible to use all three of the techniques for hint image generation. Mark all pixels unknown if they match to unknown pixels in the previous frame, or if there is an uncertainty in the match. Then, once a ternary image has been generated, erode away the foreground and background areas to enlarge the size of the unknown area.

7.4 Hint Images from Motion Fields

Given a previously generated hint image for frame n, a new hint image could be generated for frame $n+1$ by processing the motion field between frame $n+1$ and frame n.

Fig. 7.2 shows a plot of the motion vector field between frame $n+1$ to frame n of the *Dragonheart* sequence. The vector field represents the correspondence between a pixel (x, y) in frame $n+1$ to a pixel $(x+\delta x, y+\delta y)$ in frame n. The arrows in Fig. 7.2 thus show how the image

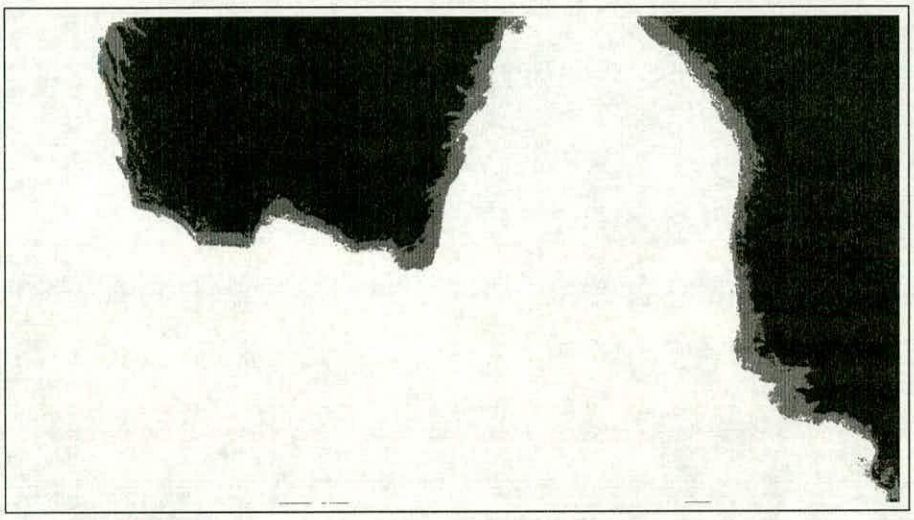


Figure 7.3: *Hint image generated using the motion field of Fig. 7.2*

changes between the two frames. (The arrows are ten times longer than the actual motion vector $(\delta x, \delta y)$ for clarity).

The motion field has been generated backwards (*i.e.* from frame 3 to frame 2) rather than from frame 2 to frame 3) in order to ensure that there is a motion vector for each pixel in the new image. False matches may otherwise cause some information to be missing in frame $n+1$.

Such a motion field can be used to generate a hint image for frame $n+1$ given an alpha channel for frame n, as shown in algorithm 9.

To build a hint image H for frame $n+1$ given a motion vector field M between frame $n+1$ and frame n, and an alpha channel α for frame n:

foreach pixel xy:
  let $(\delta x, \delta y)$ be the motion vector in M for xy
  let j be $\alpha_{x+\delta x, y+\delta y}$
  if $j = 0$ **set** H_{xy} to background
  else if $j = 1$ **set** H_{xy} to foreground
  else set H_{xy} to unknown
next pixel

Algorithm 9: *Using a motion field to generate a hint image*

The resultant image is shown in Fig. 7.3. This image contains very few artifacts, and needs no modification before use as a hint image to generate an alpha channel.

7.4.1 Calculating the Motion Field

The motion vector optical flow was generated using Black and Anandan's Robust Incremental Optical Flow algorithm [133, 134], an implementation of which is available from Black's website [135]. This code has been used because, unlike many other implementations of optical flow, it can detect large flows. Even so, the code failed with certain frames of this sequence, and with all frames of the *Teddy* sequence. It is likely that the motion was too large for the tracking to be successful.

Alternatively, a Block Matching Algorithm could have been used to generate the flow, since it can tolerate larger amounts of inter–frame movement.

7.5 Problems with Motion Vector Field Segmentation

7.5.1 Large Movements and Deformations

In special effects processing, the amount of inter-frame movement can be high. This is due to the high resolution and slow framerate of movie sequences images: an object taking two seconds to cross the scene will move 85 pixels per frame at Motion Picture resolution. Since special effects are commonly applied to fast moving action sequences, the inter-frame movement can be even higher. Further, the foreground object often fills a large portion of the frame. Optical flow assumes that the intensity of a pixel does not change as it moves between frames, which is unlikely to hold in the case of fast moving and deforming objects. Many implementations assume very small inter-frame movement. Black's Optical Flow was used because it is tolerant to relatively large inter-frame movements, but still failed with larger movements.

Large movements also present problems to block matching algorithms. If the search aperture is set large enough, the motion can be detected, as long as the block remains similar to the original. Where the object is not simply translating, but also deforming or rotating, the block being searched will change. A large search aperture is more likely to produce false matches (since there is a larger area in which a false match can be found).

Motion blur may also cause false matches. Since the amount of blur is proportional to the speed of an object across the frame, the block will appear different between frames if the object is accelerating, as between frames 2 and 3 of the *Teddy* sequence (Fig. A.4).

7.5.2 Computational Requirements

In the presence of large frames and large movements, both optical flow and block matching run very slowly. The optical flow field of Fig. 7.2 took 66 minutes to process on a 700MHz processor. A naïvely coded block match on the *Teddy* sequence between frames 2 and 3, where the interpixel motion is approx. 180 pixels, would take approximately one year using a 40 by 40 pixel block size.

The technique proposed by Mitsunaga *et al* could be adopted to increase the speed by several orders of magnitude: Recall that they used block matching to generate a correspondence for just a few points within the edge area. The remainder of the edge area was produced by drawing lines between the located points, and then marking the entire enclosed area as foreground. Such edge based techniques do not perform well where the object is not rigid. An extreme case of this can be seen between frames 2 and 3 of the *Teddy* sequence - there is a small patch of background visible between the side of the head and the ear. As the bear rotates, this patch disappears. The edge of the background patch has no correspondence in frame 3 and the technique of Mitsunaga *et al* fails.

7.5.3 Occlusions and Disocclusions

Occlusions and disocclusions cause the technique of Mitsunaga *et al* to fail, but they cause problems for all motion field techniques. No valid correspondence vectors exist for objects that are present in frame $n+1$ but not present in n. Recall that optical flow has been applied in reverse to make sure that there is a motion vector for each pixel in the new frame. Where an object enters the scene or becomes disoccluded in frame $n+1$, the motion field will contain poor data.

Occlusions (where an object present in frame n disappears in frame $n+1$) are less of a problem. At the expense of doubling the computation time and requiring extra hand–drawn hint images, dis–occlusions can be resolved by calculating optical flow in both directions. Thus, each dis–occlusion event that occurs processing the sequence forwards becomes an occlusion event when processing the sequence in reverse. Finding hint images using the motion vector field between frame $n+1$ and frame $n+2$ as well as between $n+1$ and n, and then combining the hint images together will therefore give a better solution, as long as all objects present in frame $n+1$ are visible in either frame n or frame $n+2$. This would require a method of deciding for each

pixel in frame $n+1$ whether to use the estimate from frame n or from frame $n+2$.

7.6 Colour-based Comparison

Since general motion vector field calculation algorithms have problems that make them unsuitable for use with motion picture sequences, an alternative approach has been used. Pixels in frame $n+1$ are assigned to foreground or background depending on whether their colour is most similar to the colours of foreground or background in frame n.

This section investigates a series of algorithms that generate hint images using a colour-based comparison in an attempt to provide a robust algorithm that operates quickly while still producing accurate results, *i.e.* without producing errors in the hint image which cannot be removed by the post-processing step described in the next section, and which will create artifacts in the hint image. A test image is used to illustrate the problems produced by each algorithm.

The algorithms are based on the following assumptions:

Assumption 1: within every search aperture used in every frame, the set of pixels that are part of the foreground and the set that are part of the background are separated in colourspace.

Assumption 2: Any changes in illumination conditions between frames n and $n+1$ are such that the foreground pixels in frame $n+1$ will be more similar in colour to foreground pixels than background pixels in frame n, and *vice versa*.

Assumption 3: Pixels in any disoccluded area of foreground present in frame $n+1$ but not present in frame n will be closer in colour space to foreground pixels within the search aperture than to background pixels, and *vice versa*.

7.6.1 Simple Colour Distance Classification

The basis of this algorithm is that of assumption 1 above: Measure the distance in colour space between the candidate pixel p_q and every pixel within the search aperture in the previous frame. The algorithm is as outlined in Algorithm 10.

This is a very simple colour based Nearest Neighbour Classifier [79] — it separates the reference frame into unknown, foreground and background classes and decides which of these

To classify a pixel p_{ij} in frame $n+1$ of an image sequence, with a search aperture given by a box $((x_{min}, y_{min}), (x_{max}, y_{max}))$:
set $\boldsymbol{p}$ to be the colour of pixel p_{ij} in frame $n+1$
set $d_f = d_b = d_u = \infty$
foreach pixel q_{xy} in box $((i + x_{min}, j + y_{min}), (i + x_{max}, j + y_{max}))$
  set $\boldsymbol{q}$ to be the colour of pixel q_{xy} in frame n
  set $d = \|\boldsymbol{q} - \boldsymbol{p}\|$
  if $d < d_f$ **and** q_{xy} is foreground in frame n
    set $d_f = d$
  if $d < d_b$ **and** q_{xy} is background in frame n
    set $d_b = d$
  if $d < d_u$ **and** q_{xy} is unknown in frame n
    set $d_u = d$
next q_{xy}
if d_f, d_b, d_u is the smallest of the three values, **set** pixel p_{ij} to foreground, background or unknown, respectively

If no background pixels are present within the search aperture, classify pixel as foreground and *vice versa*

Algorithm 10: *Simple colour-based hint image generation*

classes is closest to pixel $\boldsymbol{p}$ in colourspace. The unknown class need not be used if dilation is being used to generate the unknown area, which may significantly increase execution speed.

7.6.2 Increasing the Speed of the Algorithm

There are two approaches that can be employed in order to increase significantly the speed of this algorithm. Algorithms for these improvements are outlined below:

7.6.2.1 Pre-calculating the Distributions

It is often the case that the search area is extremely large, containing several thousand pixels. For every pixel p in frame $n+1$, the distance from the colour vector $\boldsymbol{p}$ to such a set of pixel colour vectors in frame n must be measured. For neighbouring pixels, almost the same area of image will be used. Significant speedup can be achieved by creating an efficient approximation of pixels within the search area in frame n, so that subsequent neighbouring pixels require fewer calculations.

The principle of this technique is to calculate the bounding box of the pixel colour vectors, and measuring the distance of pixel p to that box.

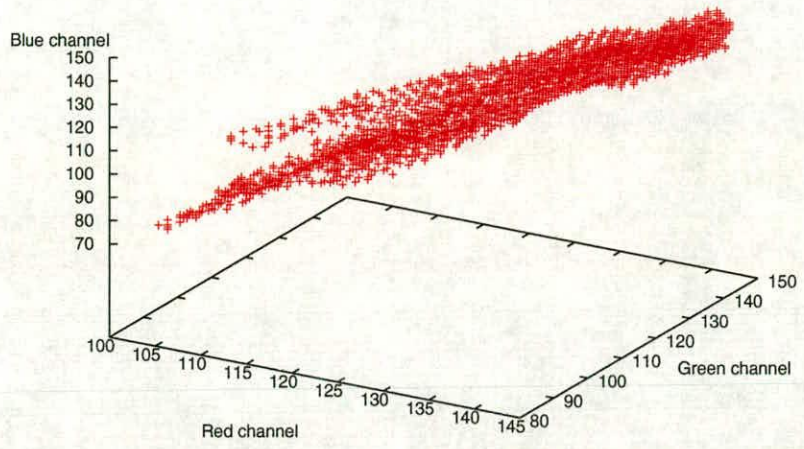


Figure 7.4: *Distribution of colour in a* 256×256 *block of an image*

Fig. 7.4 shows a point cloud of colour values for a 256×256 block of a frame of the *Teddy* image sequence. As expected, the cloud is approximately a prolate shape. Thus, an orthogonal bounding box does not fit the points well - a point close to one corner of the box would be considered close to the set because it is close to the bounding box, even though it could be a long way from any of the points. Instead of using a single bounding box, the cloud can be approximated using a set of bounding boxes (called here a *bounding set*), as seen in Fig. 7.5. Measuring the distance between $\boldsymbol{p}$ and a set of bounding boxes simply requires measuring the distance between each bounding box and $\boldsymbol{p}$. This approach is particularly useful because it gives a distance of "zero" if the point in frame $n + 1$ is inside any of the bounding boxes. Algorithm 11 builds a bounding set for a given distribution.

In order to calculate the Nearest Neighbour for a pixel $\boldsymbol{p}$ using this method, the image is split into an array of square sub–images. Bounding sets for each of these squares in the image are pre-calculated. Three bounding sets are required for each square, one for each of background, foreground and unknown area. To classify an individual pixel, the square sub–images that are covered by the search aperture are extracted and the corresponding bounding sets used.

A fast but inaccurate approach of divide the image is to use square sub–images of equal sizes. If squares are of size 128 are used, then all pixels in a 128×128 square would use exactly the same

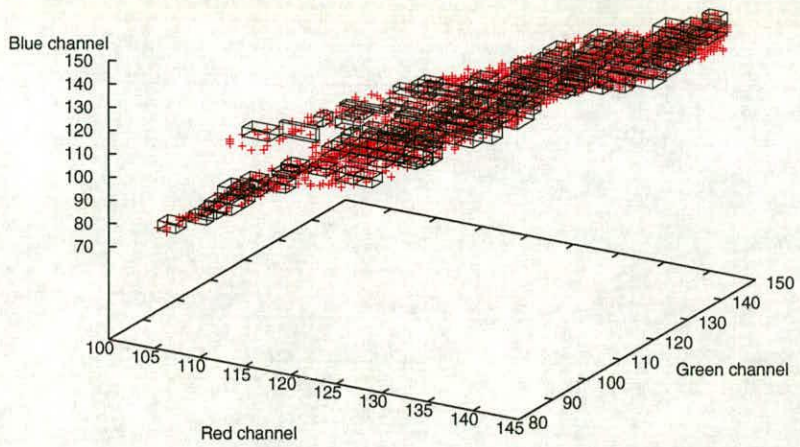


Figure 7.5: *Distribution estimated as a collection of boxes*

To build a bounding set for a cluster of pixels W

set B=BOUNDBOX $[(-\infty,-\infty,-\infty),(\infty,\infty,\infty)]$
call function SUBDIVIDE with initial bounding box B

Function SUBDIVIDE splits bounding box B into smaller bounding boxes, working recursively, and adds the boxes into the bounding set:

split $B[(r_{min},g_{min},b_{min}),(r_{max},g_{max},b_{max})]$ into eight sub-blocks, each of equal size $((r_{min}+r_{max})/2,(g_{min}+g_{max})/2,(b_{min}+b_{max})/2)$
set flag m to "unchanged"
foreach subblock S
  set S'=BOUNDBOX(S)
  if S' very small or empty
    discard S
  else
    if difference in volume between S and S' bigger than threshold
      set m "changed"
      set $S \Leftarrow S'$
    endif
  endif
next subblock S
if m is "unchanged"
  add B to bounding set
else
  foreach (possibly modified) subblock S
    call SUBDIVIDE(S)
  next S
endif

Function BOUNDBOX(B) returns the best bounding box for all points in set W bound by B
set $F=[(\infty,\infty,\infty),(-\infty,-\infty,-\infty)]$
foreach point p in W
  if p is within B
    update F to contain B
  endif
next p
return F

Algorithm 11: *Calculating a bounding set for a cluster*

group of sub–images, and so the same search area. This quantisation causes the search area to be bigger than necessary in most cases, which may cause false matches if the background and foreground are similar in places.

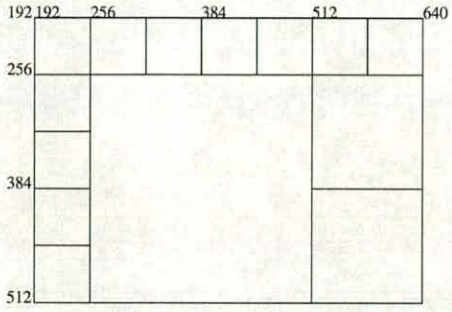


Figure 7.6: *A search aperture can be approximately represented by a selection of squares of different sizes, for each of which the bounding set has been previously calculated*

A slower but more precise approach is to calculate squares of multiple sizes (for example of size 64×64, 128×128 and 256×256). Classifying a single pixel requires finding the smallest set of squares that most accurately covers the search aperture. To save time and memory usage, the bounding set for a given square is calculated only when it is required, and discarded again when the set is no longer needed. Fig. 7.6 shows how squares may be selected to cover the search aperture. Using this technique, the search area can be modelled more accurately in order to minimise problems caused by parts of the foreground resembling parts of the background, but is slow to calculate, since each pixel will be included in squares of multiple resolutions.

Once the bounding sets that represent the search aperture have been found, the distance is calculated between $\boldsymbol{p}$ and each bounding box in each bounding set corresponding to these squares. $\boldsymbol{p}$ is assigned to whichever set corresponds to the smallest distance. In the event of a tie (typically where more than one set gives a distance of zero) the pixel is marked unknown in the hint image.

7.6.2.2 Recursive Estimation

Using pre-calculated bounding sets means that the number of distance measurements taken in order to measure each pixel p is dramatically reduced. A further speed increase can be achieved by exploiting the similarity of local pixels. The neighbours of pixels p are likely to have similar colours, and therefore likely to end up in the same classification. Ideally, neighbouring pixels

will be assigned to different clusters only on the boundary between background, foreground and unknown. Thus, pixels can be classified in groups, and subdivided only if the group is not all successfully assigned to the same group. Algorithm 12 is a recursive algorithm that implements this technique.

To build a hint image for frame $n+1$ from frame n of size (w, h)

set $t = 128$ (size of squares examined in frame $n+1$)
store bounding sets for squares of size t in frame n
for $x \in t, 2t, 3t, \cdots, w$
  for $y \in t, 2t, 3t, \cdots, h$
  call function ASSIGN(x, y, t)
  next y
next x

Function ASSIGN (x, y, t) attempts to classify a square block of width t

set bounding box B to enclose the colours of every pixel in frame n within area of image $(x, y), (x+t, y+t)$
set T to be the set of squares best covering the search aperture
set $s_{\text{foreground}} = s_{\text{background}} = s_{\text{unknown}} = \infty$
set $l_{\text{foreground}} = l_{\text{background}} = l_{\text{unknown}} = \infty$
foreach $a \in$ {foreground,background,unknown}
  set R to be the set of bounding boxes in T for area a
  foreach bounding box $E \in R$
    set $d =$ distance from near side of B to nearside of E
    if $d < s_a$ **set** $s_a \Leftarrow d$
    set $d =$ distance from far side of B to nearside of E
    if $d < l_a$ **set** $s_a \Leftarrow d$
  next E
next a
if there is a member of l that is smaller than every member of s (*i.e. all points in bounding box are closer to one region than they are to any other*)
  assign all pixels in $((x, y), (x+t, y+t))$ to smallest member of l
 else if $t > 1$
  set $t \Leftarrow t/2$
  call ASSIGN with each of $(x, y, t), (x+t, y, t), (x, y+t, t), (x+t, y+t, t)$
 else mark point (x, y) unknown
endif

Algorithm 12: *Building a hint image using recursive classification and bounding sets*

Figure 7.7 shows the result of running the bounding set algorithm on the Dragonheart sequence. The reference images are split into 128×128 squares. Each pixel is classified using the nine nearest blocks to each pixel (*i.e.* bounding sets for all pixels within a 384×384 pixel area are

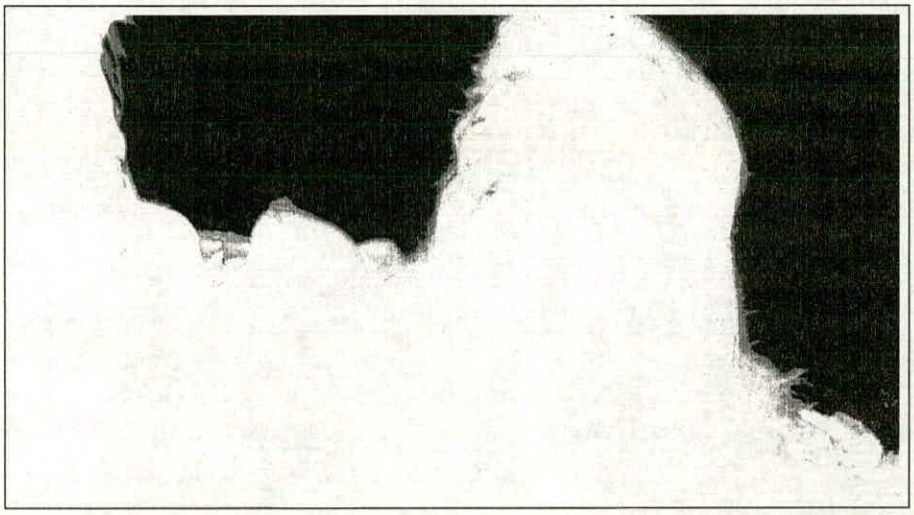


Figure 7.7: *Hint image created by the bounding set algorithm for frame 4 of the* Dragonheart *sequence, using frame 2 as the reference frame. Note the areas of foreground incorrectly classified as foreground around the head area and by the saddle of the horse*

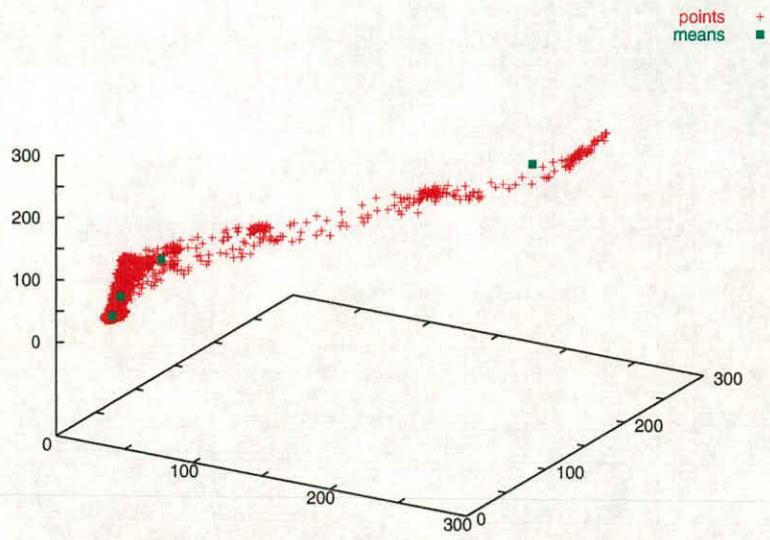


Figure 7.8: *A cluster of points approximated as four means (8 bit channel intensities)*

used for each pixel)

This algorithm produces reasonably accurate results and is able to cope with large motions and deformations, but is not much faster than optical flow, running in approximately 43 minutes per frame for the 2286×1224 pixel *Dragonheart* sequence[1].

7.6.3 Colour Distance–to–Means Classification

Bounding sets provide a precise model of the foreground and background clusters, but the overall runtime is still too slow to be practical. The distributions of each square of the reference image can be modelled faster by quantising them into several means.

Figure 7.8 shows how a cluster of points is represented as a small number of means. Orchard–Bouman quantisation [20] is used to estimate these means. To classify each pixel p with colour $\mathbf{p}$, the difference between $\mathbf{p}$ and each mean in each of the foreground, background and unknown clusters is found, and p assigned as foreground, background or unknown according to which

[1]Timings for all algorithms in this chapter are quoted for a 700MHz Duron PC with 768MB of memory, running Linux and compiling using the g++ compiler

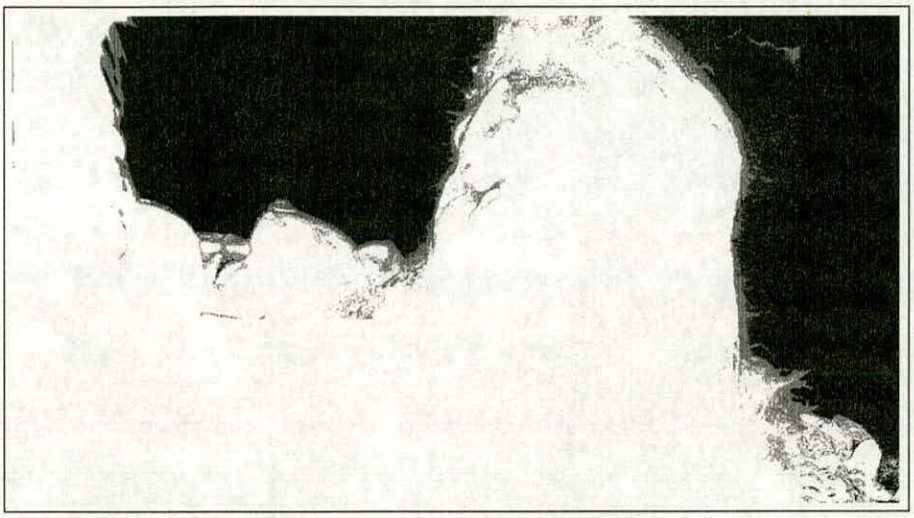


Figure 7.9: *Hint image created by the distance–to–means algorithm for frame 4 of the* Dragonheart *sequence, using frame 2 as the reference frame. The patches of background in the foreground area are more numerous and larger, and there are areas of unknown pixels in the top right of the background.*

mean in closest to p.

Fig. 7.9 shows the results of applying this algorithm to the *Dragonheart* sequence. The same block size and search area has been used as before, (384×384 search area formed from 128×128) blocks. Each block is quantised into 4 clusters.

This algorithm is the fastest presented here, running in 2 minutes, 6 seconds. Profiling the time spent by the program in each function reveals that quantising a cluster of pixels into subclusters is much faster than calculating bounding sets. Also, calculating distances between a few subcluster means is faster than calculating distances between the large number of bounding boxes required to represent the clusters.

7.6.4 Colour Probability Classification

If the variance of a subcluster is high, the means will not model closely the distribution of the cluster. Instead of measuring distances in colour space, this algorithm estimates the probability that a pixel will belong to either cluster.

To classify a group of pixels, a search aperture is scanned, and a set of background, foreground and unknown pixels extracted as with the previous algorithms. Each of these clusters is then

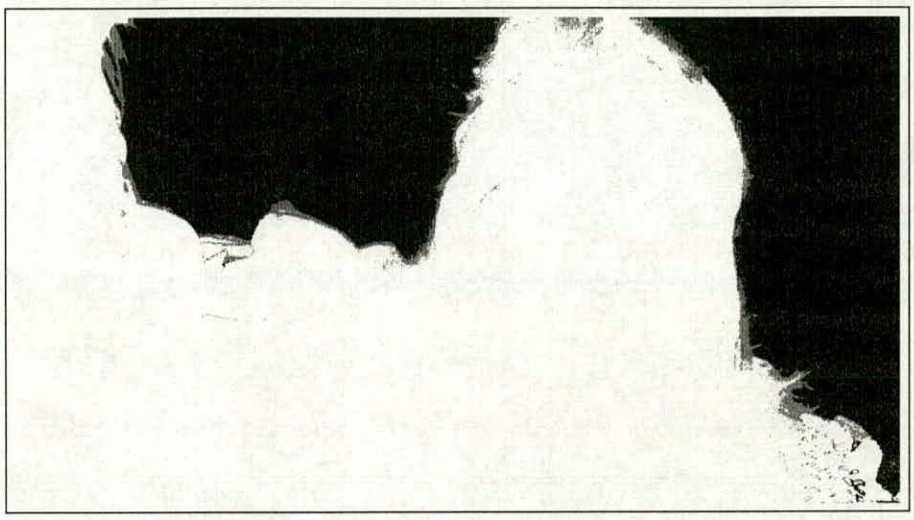


Figure 7.10: *Hint image created by the probability–based algorithm for frame 4 of the* Dragonheart *sequence, using frame 2 as the reference frame*

quantised into subclusters and approximated as a set S of orientated Gaussian ellipses (as for the Genetic Algorithm of section 6.5) The probabilities $p(f)$,$p(b)$,$p(u)$ of each pixel p belonging to the foreground, background or unknown areas respectively is given by

$$p(f) = \frac{1}{\omega|F|}\sum_{j\in F} N_{F_j}\left|\Sigma_{F_j}^{-1}\right| e^{\frac{-(P-\overline{F_j})^T\Sigma_{F_j}^{-1}(P-\overline{F_j})}{2}} \tag{7.1}$$

$$p(b) = \frac{1}{\omega|B|}\sum_{j\in B} N_{B_j}\left|\Sigma_{B_j}^{-1}\right| e^{\frac{-(P-\overline{B_j})^T\Sigma_{B_j}^{-1}(P-\overline{B_j})}{2}} \tag{7.2}$$

$$p(u) = \frac{1}{\omega|U|}\sum_{j\in U} N_{U_j}\left|\Sigma_{U_j}^{-1}\right| e^{\frac{-(P-\overline{U_j})^T\Sigma_{U_j}^{-1}(P-\overline{U_j})}{2}} \tag{7.3}$$

$$1 = p_f + p_b + p_u \tag{7.4}$$

where F, B, U are the clusters formed from the foreground, background and unknown areas respectively, $|X|$ is the total number of pixels in the cluster X and $\overline{X_j}$, Σ_{X_j} and N_{X_j} are the mean, covariance matrix and size in pixels, respectively, of subcluster j of X. ω is a weighting factor to ensure that the total probability is 1. The pixel is then assigned to the cluster which has the highest probability.

The result of applying this algorithm to the *Dragonheart* sequence is shown in Fig. 7.10. This

contains much less noise in the background and foreground area than the pure distance–based algorithms, but the unknown area is much thinner than it should be. The distance–based techniques perform better in this area.

This algorithm runs in 2 minutes 32 seconds, slightly slower than the distance–to–mean algorithm. Since the exponentials give very small numbers (approximately 10^{-200}), extended accuracy floating point precision (C `long double` variables) are required, which are slow to compute. Calculating the inverse of the covariance matrices also requires many floating point multiplications.

7.6.5 Summary

This section has presented four algorithms. The colour distance classifier runs far too slowly to be practical, and the distance–to–means algorithm produces too many errors which will be apparent in the final alpha channel. Of the remaining algorithms, the bounding box algorithm tends to produce hint images which are less prone to errors, but is much slower than the Maximum Likelihood, probability based algorithm. This probability based algorithm will therefore be used for the remainder of the chapter.

7.7 Noise Removal

All of the results shown in the previous section contain small amounts of noise: Small groups of pixels have been classified as unknown instead of background or foreground. If these images are used as hint images directly, some of these pixels will cause noise in the final alpha channel.

Fig. 7.11 shows how these pixels are misclassified. The pixel colour vectors within a small area of frame 1 of the *Teddy* image sequence are plotted. The area covers background, foreground and intermediate pixels (as defined in the calculated alpha channel), and Fig. 7.11 colour codes these regions accordingly. Clearly, there is significant overlap between the intermediate region and both the foreground and background clusters. Any pixel from frame 2 of the *Teddy* sequence whose colour is in the area where the unknown and background overlap will be classified either as intermediate or as background, depending on the classification of the nearest pixel. It is this effect that causes pixels to be incorrectly classified in the results of the *Dragonheart* images.

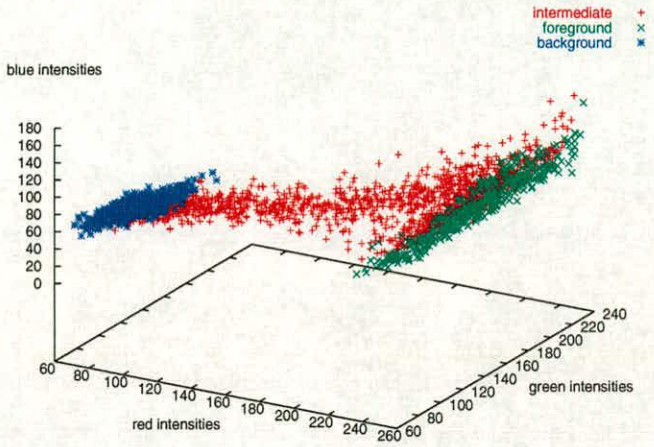


Figure 7.11: *A small area of frame 1 of* Teddy *sequence, showing the classification of each pixel*

In the case of the two distance–based algorithms, a K Nearest Neighbour (KNN) Classifier [79] could be used instead of the simple nearest neighbour classifier described previously. Rather than setting the classification of the candidate to be the same as the nearest mean, or the nearest bounding box, the classification of each of the nearest K pixels is found, and the candidate set to be the same class as the majority of the K nearest boxes or means. (K is odd to prevent ties). This is useful to eliminate problems caused by a single outlier between two otherwise separated clusters, but is less helpful where there is significant overlap. (*i.e.* where there are more than K pixels in the overlapping area). Furthermore, KNN is more computationally intensive, as a list of K pixels needs to be maintained in sorted order. The use of KNN instead of standard single Nearest Neighbour has been abandoned since its inclusion in [136], in favour of the post-processing technique outlined here.

7.7.1 Small Region Removal

Rather than attempting to develop a more robust classifier to prevent noise in the resultant image, the noise in the result is removed separately. There are various techniques that can be used to remove small amounts of noise in images: Morphological closure and median filtering will remove most of the noise, but will corrupt the edges in the images.

The approach taken here is that described by Salembier *et al* [137]. This works by converting

To remove regions smaller than size t from an image W
let I be the REGION IMAGE of W
let c be the number of regions in I
let R be the REGION ADJACENCY GRAPH of X
let T be the MAX-TREE of X using Y
let E be an array of size c, where each $E_i = i \forall i(0, c)$
let N be an array of size c, where each N_i is set to intensity of pixels in region i

repeat
  foreach node n in T with a single region in n_R
  let s be the total size of all regions in descendent nodes of n
  if $s < t$ (descendents of n smaller than threshold)
    foreach region c in subtree from n
      set $E_c \Leftarrow r$
    next region
    remove all descendent nodes from n
  endif
until no nodes have been merged this step
foreach pixel ij in image
  let $t = I_{ij}$ (region number of ij)
  let $r = E_t$ (region that replaces t)
  set $W_{ij} = N_r$ (intensity of region that replaces r)
next pixel

Algorithm 13: *Small region removal algorithm*

the hint image into a collection of regions, and then merging regions together if they are too small to be anything other than noise. There are two areas of novelty in the approach outlined below. Because the images are so large, a region adjacency graph [68] is used to store information about connectivity of regions (Salembier *et al* threshold and re-region grow in order to detect connectivity). Secondly, using a region image to form the Max-tree, rather than the original image, permits removal of black noise from a white area, which is otherwise impossible.

To build a REGION IMAGE X of image W
mark all pixels in X as zero
set $c = 1$
repeat
  scan for next pixel p_{ij} in X marked as zero
  region grow from p, setting $X_m = c$ for all m in the region
  set $c \Leftarrow c + 1$
until no pixels in X are marked as zero

Algorithm 14: *Creating a Region Image*

The novel approach is defined in detail in Algorithm 13. This algorithm first generates a region image using Algorithm 14. This image is then converted into a Region Adjacency Graph using

To make a REGION ADJACENCY GRAPH R from region image X

let a Region Adjacency Graph have, for each node n:
    a region number n_r
    a set of edges E to other nodes

let R be a graph with one node n for each region in X, with no edges
foreach pixel q in image X
  let p be the region number of pixel q
  if a neighbour n of q is in a different region r to p
    if no edge exists from node p to node r
      insert edge from node p to node r
next pixel q

Algorithm 15: *Creating a Region Adjacency Graph*

To make a MAX-TREE T from region adjacency graph R
let a Max-tree have, for each node n :
    A set of regions n_R
    A set of child nodes n_C

let T be a single node tree where n_R contains every region
while there exists a node n in R, with more than one region in n_R
remove all regions except the one with the lowest index from n_R and store in set S
foreach region r in S
  foreach child node c of node n
    if r is connected to any region in c_R (*i.e.* edge in RAG)
      insert region r into c_R
    endif
  next child node
  if r not inserted **or** no children exist
    create new node child node in n_C containing r
endwhile

Algorithm 16: *Building a Max-tree using a Region Adjacency Graph*

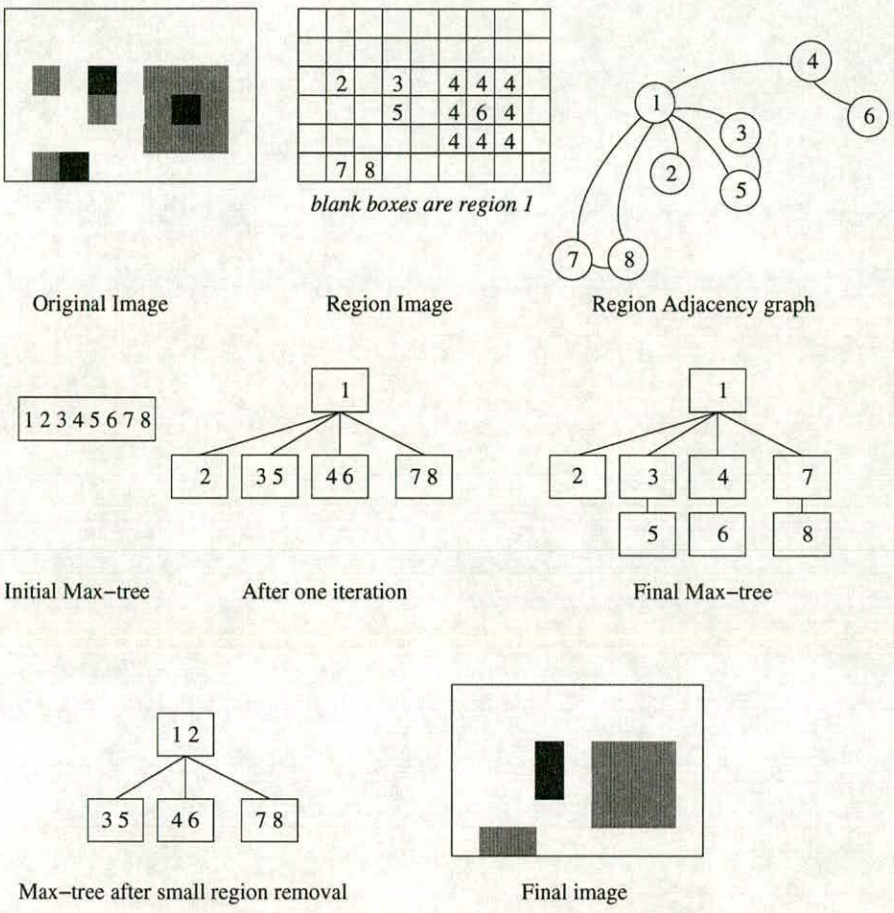


Figure 7.12: *Max-tree noise removal in a sample image*

Algorithm 15. Nodes in this graph represent each individual region in the image, and the connections between them indicate which regions are directly adjacent in the image. A Max–tree is produced using 16. The root of this tree is the first region found in the image, and the individual branches represent groups of connected regions in the image. Algorithm 13 proceeds by merging nodes in the max-tree which represent regions that are too small to exist into their parent nodes, and updating the colour of the region to be the same as the parent. Fig. 7.12 shows the data structures used with a small sample image. In this image, region six is a child node of region four, since it is surrounded by it in the original image. Region six is removed by merging it with region four, and all pixels within it are recoloured the same shade as the parent region.

Fig. 7.13 shows the result of applying this algorithm to Fig. 7.9. All regions smaller than 1000 pixels are removed.

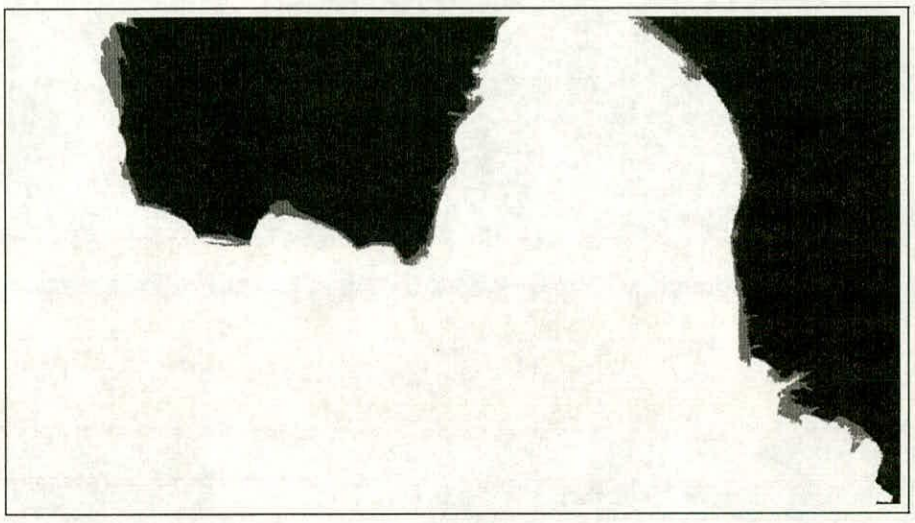


Figure 7.13: *Fig. 7.10 after applying small region removal*

7.7.2 Region Voting

Fig. 7.9 was produced by removing small regions from the hint image. An alternative approach is to use regions in the original image. If the colour distributions of background and foreground are sufficiently different, a threshold can be found that partitions the input frame into regions such that no region covers both background and foreground. Fig. 7.14 shows the result of partitioning frame 4 of the *Dragonheart* sequence into multiple regions. Again, all regions smaller than 1000 pixels are removed.

Noise can be removed from the hint image by applying a region voting technique. Since each region in the input belongs to the same region (background or foreground), every pixel in the area corresponding to the region should be in the same class. This is enforced by applying a voting strategy. For each region in the partition of the input frame, the number of foreground, background and unknown pixels in the hint image is found. Each pixel in the region is then set to be the same as whichever of these is the greatest.

Fig. 7.15 shows the resultant hint image. Since the region partition of the input frame tends to eliminate the unknown area (the regions produced are small and hence merged with background or foreground), this image has almost no unknown pixels. A further disadvantage of region voting is the time required to run. The original region image for the frame contained over 16,000 regions which was reduced to 250 regions using the small region removal algorithm.

Figure 7.14: *Frame 4 of the* Dragonheart *sequence partitioned into regions. Each region has been assigned a different random colour*

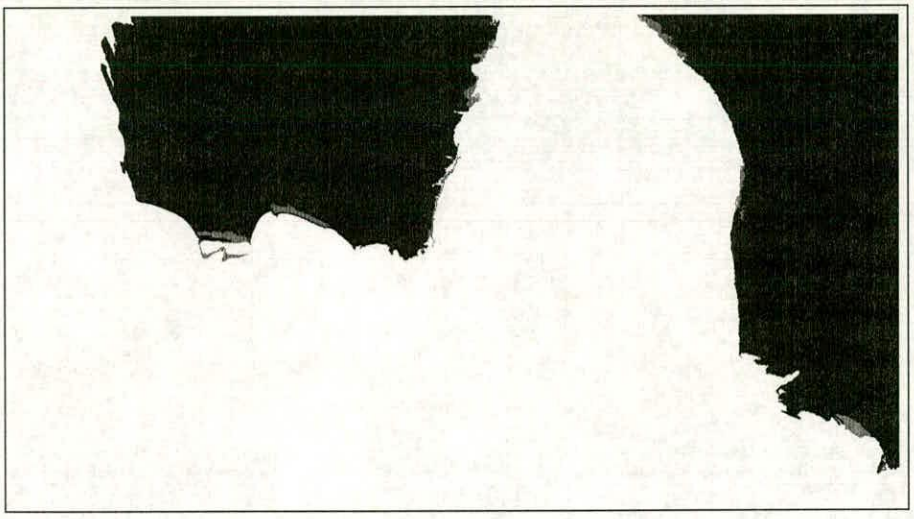


Figure 7.15: *Fig. 7.10 after applying region voting*

For the results presented in this thesis, small region removal in the hint image was used to reduce noise in the hint image. However, region voting may be a useful technique if the results are particularly noisy.

7.7.3 Dilation

The colour based algorithms tend to underestimate the size of the unknown area, marking pixels that should be unknown as foreground or background. The noise removal algorithm also causes the unknown area to become thinner if it is made up of small disjoint regions. The simplest solution to this problem is to dilate the unknown area in the image by a small amount. Any pixel that is marked as background or foreground and is within a small distance (typically four or five pixels) of an unknown pixel is also marked as unknown.

7.8 Progressive and Non–Progressive Estimation

The algorithms presented here have described hint image generation for frame $n+1$ from frame n. Each frame could be generated either from its previous frame (*progressive estimation*) or all frames could be generated from the same reference frame (*non–progressive estimation*). The previous frame will be more similar to the current frame than the reference frame, and the inter–frame movement will be smaller, but any errors in the alpha channel generated for frame $n+1$ will propagate through to frame $n+2$.

In this thesis, non–progressive estimation has been used. Progressive estimation requires the generation of both the hint image and alpha channel for the previous frame before calculation of the hint image for the next frame can begin. With non–progressive estimation, each frame depends on only the source frame and is independent of other frames. If there is enough compute power available, segmentation of every frame of the sequence can begin in parallel as soon as the alpha channel of the reference frame has been generated. This requires at least one processor per frame[2]. Computationally intensive tasks such as image rendering are often run on a large network of separate computers, such as Linux boxes. The rendering for the *Toy Story* films as well as many of the special effects for the film *Titanic* was achieved in this manner [138]. Such networks could easily be used for hint image and alpha channel estimation in order to

[2]The possibility of further speed increases by using more than one processor per frame is considered in section 8.4.2

render a sequence very quickly.

7.9 Multiple Reference Frames

Whether progressive or non–progressive is employed, it is unlikely that an extremely long sequence could be generated from just a single manually initiated input frame. After a few frames the foreground object is likely to have changed so much it would be too different from the reference frame. Thus, a new reference frame (or *keyframe*) must be generated every few frames. The keyframes need not be entirely hand–drawn: Suppose hint image for frame 2 has been created, and a new hand–drawn keyframe is required for frame 12. The hint image for frame 12 can be estimated from frame 2 and then touched up by hand in order to create a more accurate hint image, which is likely to be less effort than drawing the hint image from a blank image.

When estimating the hint image for a frame, it is sensible to use the nearest keyframe (whether it is before or after the frame under classification in the sequence). However, it is also possible to use *both* nearby keyframes to estimate the image. If, for a given pixel p, the estimate for p from keyframe k_0 differs for that for p from keyframe k_1, one of the results must be selected. In the case of the distance–to–mean algorithm, Algorithm 17 can be used to classify a frame using two keyframes. It is based on the ratio of the distances to the nearest two clusters. This ratio is high when p is very close to one cluster but further from the others, and low where it is in between two clusters or where the clusters are close together. In the latter case, the estimate is less reliable than the former, so choosing the result from the reference frame with the highest such ratio is likely to select the best result.

To assign a value to pixel p (with colour **p**) using two keyframes k_0 and k_1:
let $d_{k_i 0}$, $d_{k_i 1}$ and $d_{k_0 2}$ be the distances in colour space between **p** and the nearest mean in the background, foreground, and unknown clusters in keyframe $i \in \{0,1\}$ respectively.
for $i \in \{0,1\}$
  set a_i to the number of the cluster in the set $\{d_{k_i 0}, d_{k_i 1}, d_{k_0 2}\}$ with the smallest value
  let v_i be the value of the smallest of $d_{k_i 0}$, $d_{k_i 1}$ and $d_{k_0 2}$, and w_i the second smallest.
  let $q_i = \frac{w_i}{v_i}$
if $q_0 > q_1$ **assign** p to set a_0, **else assign** p to a_1x

Algorithm 17: *Colour-distance based classification with multiple keyframes*

Extension of Algorithm 17 to more than two keyframes is trivial. In the case of probability–based classification, pixel p can be assigned to whichever cluster has the highest probability.

7.10 Alpha Channel Estimation in Motion Picture Sequences

Figure 7.16 shows a system for segmenting image sequences. This system serves two purposes: Firstly, it shows one way of integrating the single alpha channel estimation algorithms of chapters 5 and 6 with the hint image update algorithms in this chapter in order to segment an entire moving image sequence without the need to generate by hand a hint image for every frame of a sequence. Secondly, this system is used in section 7.11 to demonstrate and evaluate the probability based hint image update algorithm.

The system operates as follows: A hint image for the first frame is generated by hand, and used to create an alpha channel for this frame. Any of the algorithms presented in the previous chapters may be used for this process. This hint image is used to process the final frame (the only other keyframe used in this example) to produce a hint image which, after noise removal, is corrected by hand in order to produce an alpha channel for the final frame. Hint images are then estimated independently for the intermediate frames (the frames have been shown amalgamated for clarity), which can then be processed automatically to produce alpha channels for the entire sequence.

This layout is independent of the algorithms chosen: Any alpha estimation algorithm can be used to generate alpha channels, and any hint image update algorithm used to generate hint images.

The system outlined is suitable where the image sequence is varied, and multiple reference frames are required to generate hint images for each frame. Other arrangements are possible: If the sequence does not change significantly or is very short, it may be possible to generate every hint image in the sequence automatically from a single hint image and use only one reference frame. In the worse case, it may be necessary to hand modify the hint image for each frame of the sequence.

7.11 Results

Results for two different sequences are presented here in order to demonstrate the hint image estimation algorithm. These results were produced using a system similar to that described in fig 7.16. Different hint images can produce identical alpha channels and it is not always easy to see how different errors and artifacts will affect the final alpha channel. Thus, the hint image

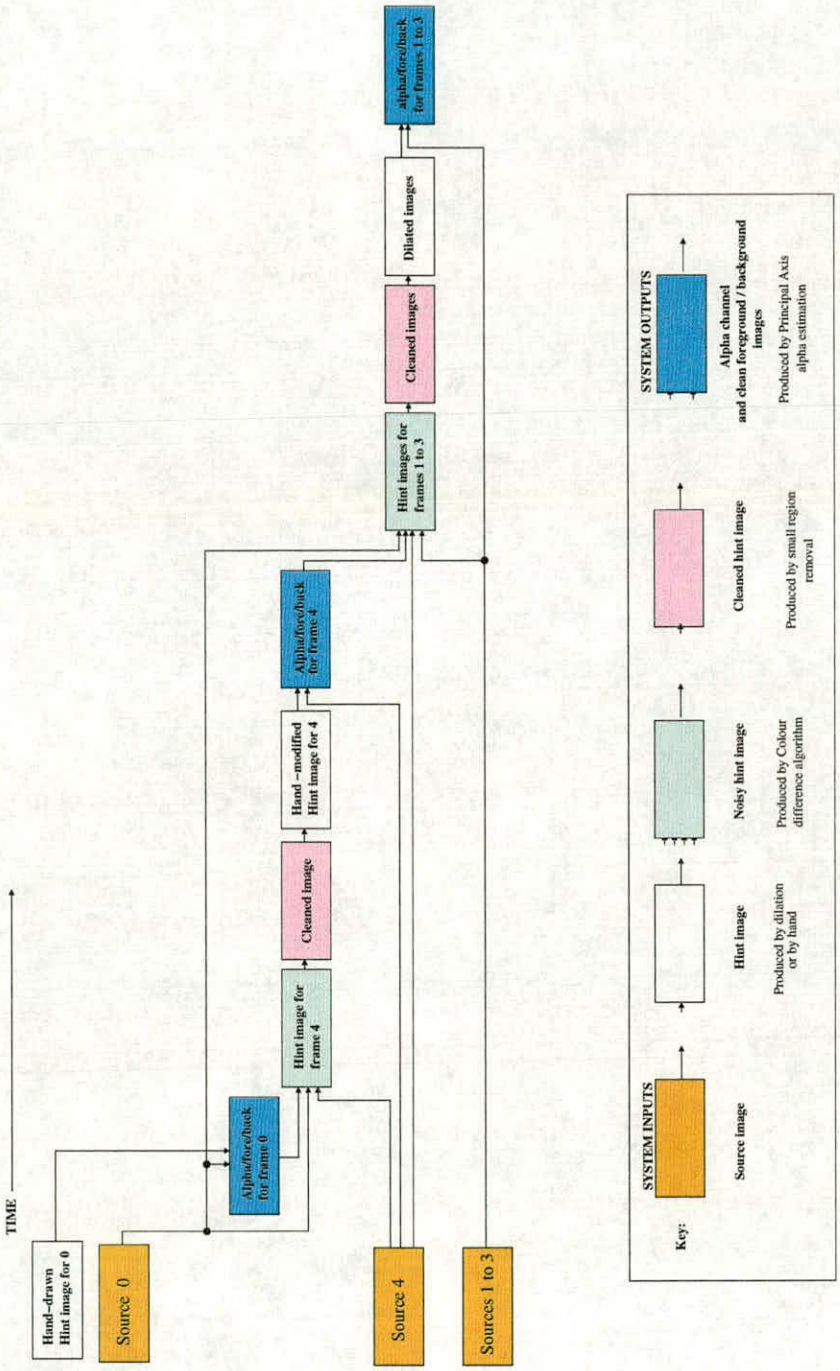


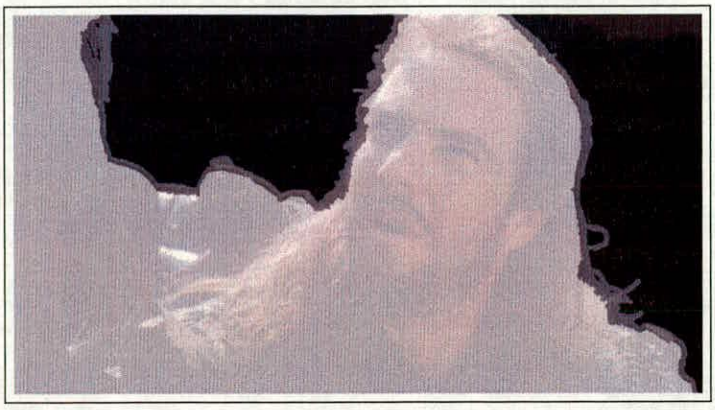
Figure 7.16: *Complete system for alpha channel estimation of motion picture sequences*

estimation algorithm must be assessed in conjunction with the alpha estimation program by examining the alpha channel produced for major artifacts, such as holes within the foreground or parts of the original background appearing within the composite image.

For both the sequences presented here, the probability based hint image algorithm is used in conjunction with the Principal Axis algorithm. These two algorithms were chosen because they are fast and produce accurate results. Other hint image update algorithms, for example, were either so slow that the *Dragonheart* sequence would have taken many days to process, or else produce very poor results.

7.11.1 The *Dragonheart* Sequence

The *Dragonheart* sequence was processed using the first and last frames of the 21 frame sequence as keyframes. Fig. 7.17 shows the original hint image and the alpha channel and composite result produced for frame 1. Frame 21 was then processed using the hint update technique. The resultant hint image 7.18(a) is too noisy to be used directly, and the noise removal algorithm is applied to produce the clean image of 7.18(b). The resultant image needs very little modification — dilation has been applied, and the saddle area is slightly modified — to produce a hint image 7.18(c). The alpha channel produced is shown in Fig. 7.19.



(a) hand–edited hint image

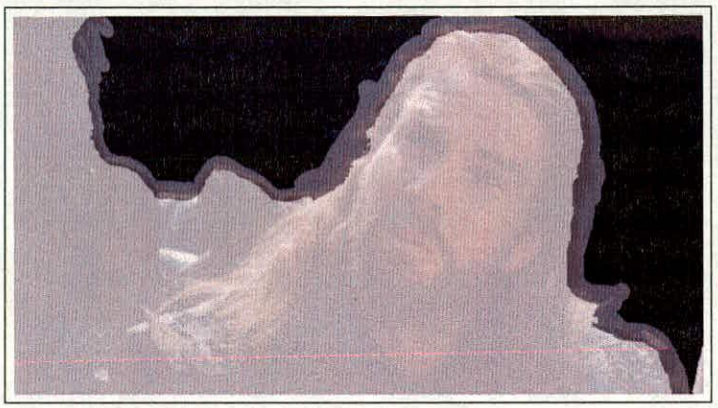
(b) resultant alpha channel

(c) Composite image

Figure 7.17: *Results for frame 1 of the* Dragonheart *sequence*

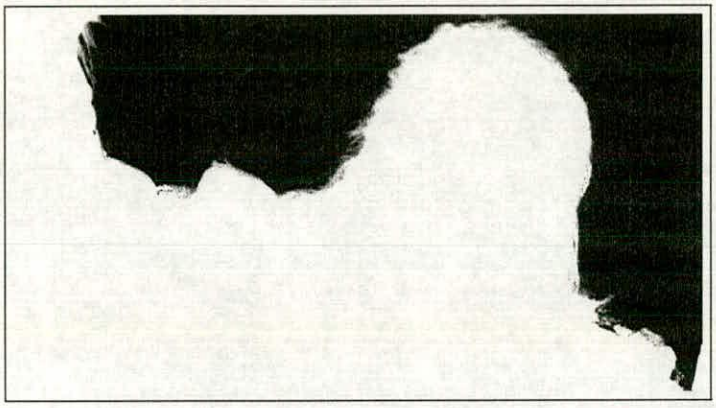
(a) Rough hint image produced from frame 1

(b) Noise removed (all regions less than 2000 pixels merged)



(c) Hand modified hint image

Figure 7.18: *Hint images produced to process frame 21 of the* Dragonheart *sequence*



(a) Alpha channel

(b) Composite image

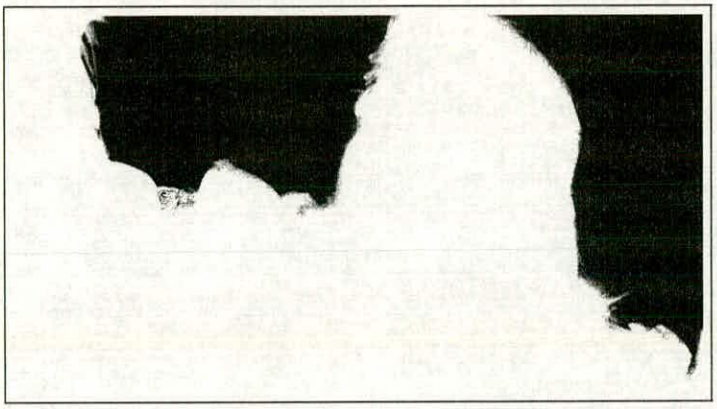
Figure 7.19: *Results for frame 21 of the* Dragonheart *sequence*

(a) Rough hint image produced from frames 1 and 21

(b) Noise removed (all regions less than 2000 pixels merged)

(c) Unknown area dilated by 3 pixels

Figure 7.20: *Hint images produced to process frame 5 of the* Dragonheart *sequence*



(a) Alpha channel

(b) Composite image

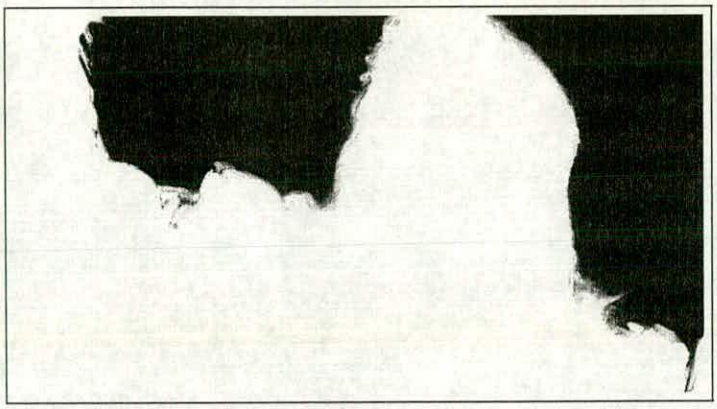
Figure 7.21: *Results for frame 5 of the* Dragonheart *sequence. Note the hole in the horse's saddle area*

(a) Rough hint image produced from frames 1 and 21

(b) Noise removed (all regions less than 2000 pixels merged)

(c) Unknown area dilated by 3 pixels

Figure 7.22: *Hint images produced for frame 10 of the* Dragonheart *sequence*



(a) Alpha channel

(b) Composite image

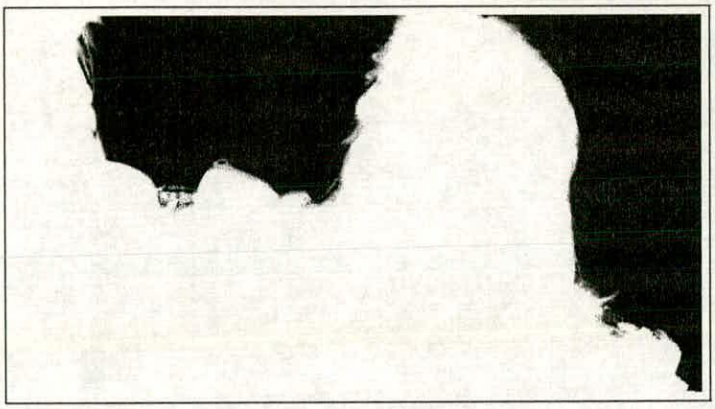
Figure 7.23: *Results for frame 10 of the* Dragonheart *sequence. Again, note the hole in the horse's saddle area, which appears more pronounced than in frame 5*

(a) Rough hint image produced from frames 1 and 21

(b) Noise removed (all regions less than 1000 pixels merged)

(c) Unknown area dilated by 3 pixels

Figure 7.24: *Hint images produced to process frame 15 of the* Dragonheart *sequence*

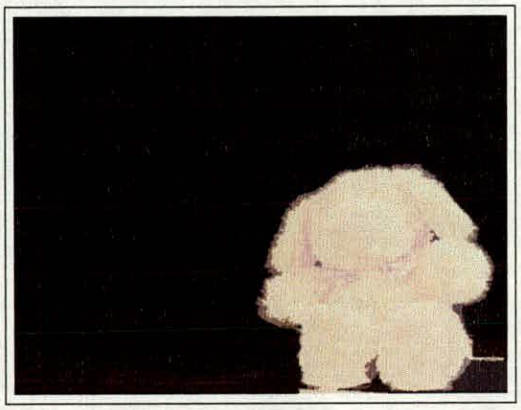


(a) Alpha channel

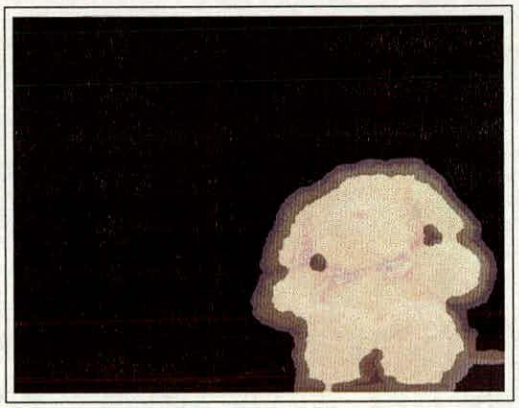
(b) Composite image

Figure 7.25: *Results for frame 15 of the* Dragonheart *sequence. The hole artifact in the saddle area appears similar to frame 10*

(a) Rough hint image produced from frame 1



(b) Noise removed (all regions less than 2000 pixels merged)

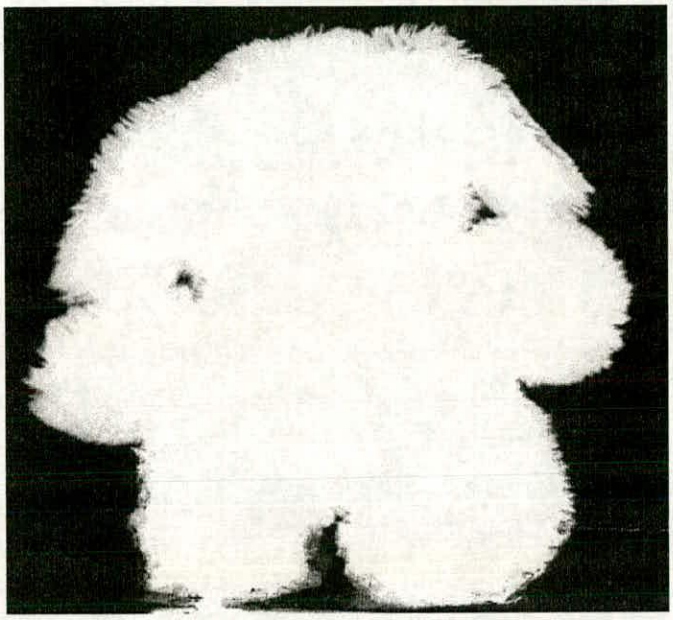


(c) Unknown area dilated by 3 pixels

Figure 7.26: *Hint images produced to process frame 2 of the* Teddy *sequence*

7.11.2 The *Teddy* Sequence

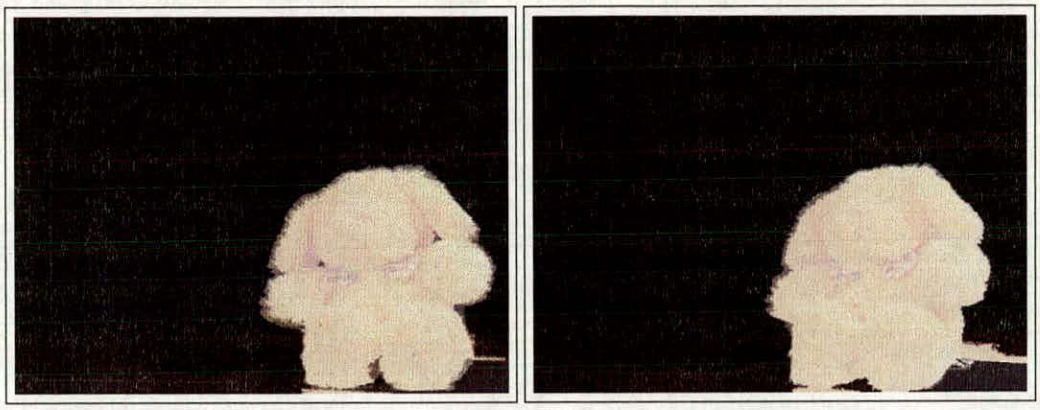
Results of processing frames 2 and 3 from frame 1 of the *Teddy* sequence are shown in figs. 7.27 and 7.29. Hint images produced by the alpha estimation step are shown in figs. 7.26 and 7.28. The disoccluded background at the right of the image is more similar to the foreground areas of frame 1 than the background area, and is classified as foreground. Since frame 2 is reasonably well segmented, better results would be achieved by using frame 2 as the reference frame rather than, or as well as, frame 1. In most practical cases, the sequence would be much longer than 3 frames, and a later keyframe would be generated. Thus, the area disoccluded between frames 2 and 3 would be marked as background in the later keyframe and so classified correctly.



(a) Alpha channel

(b) Composite image

Figure 7.27: *Results for frame 2 of the* Teddy *sequence. The artifact underneath the back right paw is caused by the alpha estimation algorithm, but the slight transparency of the left ear is probably due to overdilation of the hint image*

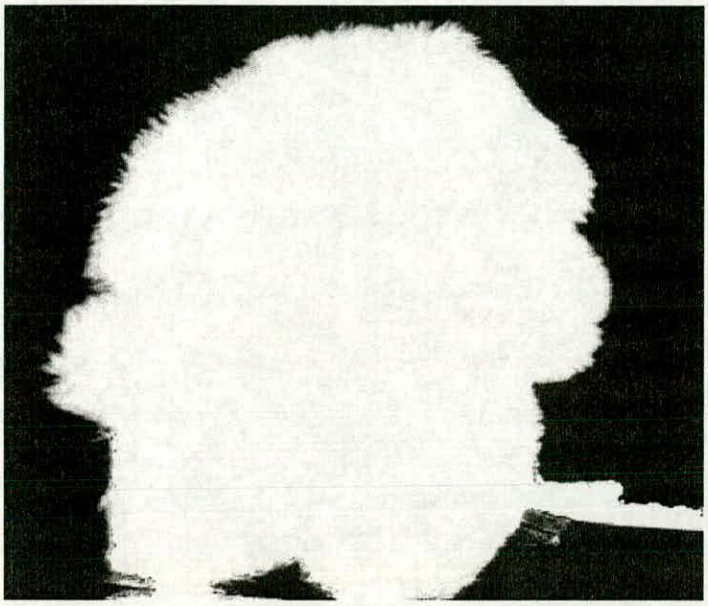


(a) Rough hint image produced from frame 1

(b) Noise removed (all regions less than 3000 pixels merged)

(c) Unknown area dilated by 3 pixels

Figure 7.28: *Hint images produced to process frame 3 of the* Teddy *sequence*



(a) Alpha channel

(b) Composite image

Figure 7.29: *Results for frame 3 of the* Teddy *sequence. Note the large area of disoccluded background, which has been incorrectly classified as foreground by the hint image algorithm and appears in the final algorithm. Transparency around the ears visible in frame 2 seems to be much reduced in this image.*

7.12 Summary

This chapter has developed a series of algorithms that perform automatic hint image updating. Optical flow, and other motion estimation techniques, occasionally perform well but are not robust against fast motion. This problem, and their slow performance, led to the development of a set of algorithms that use colour distance to classify. The novel Bounding Set representation can be used in general Nearest Neighbour classifiers, while a novel application of a Maximum Likelihood probability based classifier produces good results at high speed.

These algorithms have difficulty in producing accurate hint images in some cases, resulting in artifacts appearing in the final alpha channel. These appear as sections of foreground which should be classified as background and *vice versa*. These occur where the colours overlap between foreground and background in the reference image (as in the saddle area of the *Dragonheart sequence*) or where disoccluded background is more similar to parts of the foreground than to the background in the reference frame (as in the right hand side of the *Teddy* sequence). However, these algorithms provide a significant improvement over generating hint images by hand and, where they do not perform accurately, almost always provide results that require very little modification to correct.

Chapter 8
Conclusions

8.1 Introduction

This section is organised as follows: Section 8.2 summarises the topic and contribution of the thesis. The topic of parameter selection is considered in section 8.3. Issues that arise from the practical implementation of the algorithms presented in this thesis are considered in section 8.4. A critical discussion is presented in section 8.5. Suggestions for future work are presented in section 8.6. Section 8.7 concludes this thesis.

8.2 Summary

This thesis has presented novel techniques and novel adaptations of existing techniques in order to perform alpha channel estimation for segmentation of motion picture resolution sequences.

In chapter two, digital compositing and the concept of an alpha channel were introduced. Alpha channel estimation is a type of segmentation that involves, for each pixel in the image, estimation of "clean" foreground and background colours, a linear combination α of which will reproduce the original colour of the pixel.

Chapters three and four formed a survey of standard image segmentation techniques, both for still images and for image sequences. The techniques used in motion picture imaging tend to be fairly limited in the images that they can segment, but can produce very detailed alpha channels.

Chapter five presented a group of novel algorithms that perform alpha channel estimation on images with natural backgrounds. The algorithms require user interaction: A "hint image" must be drawn to indicate the position of some pixels that are unmixed foreground and background. Since these pixels do not require computation, this form of initialisation also reduces the amount of processing required. The most effective algorithm was shown to be the **Principal Axis Algorithm** that classifies each pixel by sampling nearby known background and foreground

colours and approximating clusters of these as a line. The algorithm operates in region growing fashion to allow it to reuse previous result.

Chapter five also presents a novel technique for performing alpha estimation in the presence of **backlighting**. Without the use of this technique, backlighting causes poor performance.

Chapter six included a survey of alpha estimation techniques that were published while work on the techniques of chapter five was in progress. The Gaussian Mixture Model approach taken by Ruzon and Tomasi and by Chuang *et al* inspired the development of a **Genetic Algorithm** based alpha estimation algorithm that classifies each pixel by sampling nearby known background and foreground colours, deriving a GMM of these, and searching for the optimal clean foreground, background and α values for these pixels using.

The results presented show that the algorithms developed by the author perform at least as well as, and in many cases much better than, alternative alpha estimation techniques. They also show that the Principal Axis algorithm executes considerably faster than any other published algorithm, and performs extremely well even in the presence of noise and where the background and foreground are fairly similar.

Chapter seven considered the application of alpha channel estimation to moving pictures. Since standard motion estimation techniques are slow and unreliable when applied to fast moving high resolution image sequences, a novel colour–based algorithm was developed. This algorithm provides extremely fast updates of the hint image in order to segment subsequent frames of the sequence. This algorithm works by assigning each pixel in the new frame to background or foreground using a Maximum Likelihood approach.

The algorithms of chapters five, six and seven are intended to operate together in the manner outlined in Fig. 7.16. Chapter seven shows the results of applying this system to two image sequences. The algorithm fails only in the case that the background and foreground colours become too similar.

8.2.1 Performance

In the test images and sequences used in this thesis, these algorithms have been shown to perform better than some or all of the previously published solutions to the problem. Either the result is more acceptable (*i.e.* more accurate) or the algorithm is more efficient.

While these algorithms are not able to segment foregrounds away from backgrounds in every situation, they require fewer restrictions than the standard bluescreen algorithms and thus allow greater flexibility when filming special effects sequences.

Since each of the algorithms discussed here — including those presented by other authors — have different strengths and weaknesses, different images are best segmented with different algorithms. No single algorithm is the best in all circumstances. Since only limited data was available it was not easy to develop a formal set of guidelines for choosing which algorithm is suitable in any given case, but the following list of "rules of thumb" may be helpful.

For alpha channel estimation:

- Where the background and foreground are well separated in colourspace, the faster Principal Axis algorithm will probably produce results little different to the other algorithms, and will run quickly.
- Where the background and foreground are highly non-uniform but still separated in colour space, the algorithms of Chuang *et al* and of Ruzon and Tomasi will produce a more acceptable result.
- Where the background and foreground are non-uniform and begin to overlap in colourspace, the Genetic Algorithm may produce the best results.
- The Principal Axis algorithm occasionally produces the best results on images with close backgrounds and foregrounds, especially if alpha range adjustment is used. This can be true even if the clusters are not prolates.

For hint image creation:

- Where the background and foreground are relatively well separated, the Maximum Likelihood algorithm produces good results extremely quickly.
- Where the background and foreground are close but not overlapping in colourspace, the Bounding Box algorithm will produce better results but requires more time.
- Where there is little movement in the sequence, Optical Flow produces a good result even if the foreground and background are overlapping in colourspace.

8.3 Parameter Selection

These algorithms all require the selection of a set of parameters in order to produce an acceptable result. Algorithms that require manually adjusted parameters are not uncommon in image processing or in Special Effects Processing. Many users prefer to have many parameters, as they may be able to tweak the output according to how they wish it to appear. The Ultimatte Bluescreen plug-in (an implementation of the Vlahos patents [44]) for Adobe's *Premiere* video editing package contains 45 sliders for internal parameter adjustment.

The requirement for a set of parameters (7 in the case of the Principal Axis Algorithm; 13 in the case of the Genetic Algorithm) is unfortunate as it makes the use of the program less automatic. Many of the parameters are intuitive. For example for an image with a smooth background, the number of pixels selected from the background can be reduced; if the foreground changes quickly away from the line drawn, the width of the stripe must be set small; if foreground and background are a separated in colourspace, the number of subclusters used to represent them can be reduced. Some of the parameters have very little effect on the final result. However, for some images, the settings of non-intuitive parameters are more important. If the algorithm could run fast enough so that the parameters can be changed and a preview of the result of the changed viewed interactively, then the reliance on several parameters is not overly problematic.

Some parameters can be estimated automatically. In the case of the hint image update algorithm, every pixel that is marked as background because it is too far away from the foreground area must be surrounded by pixels that are background. If this is not the case, the image appears blocky. This can be detected automatically and the search aperture widened.

8.3.1 Colourspace Selection

Ruzon and Tomasi selected the $CIE - Lab$ space to perform their alpha estimation. Experiments using different colour models for each of the novel techniques presented here were inconclusive. No colour model seemed better than others. In many cases for alpha estimation, the RGB colour model performed slightly better. This is not surprising. The algorithms assume that blended areas are a linear combination of background and foreground colours. Since sensors are RGB sensitive, the blending caused by, for example, motion or focal blur, will be linear only in the RGB colourspace. Since RGB performed at least as well as other colourspaces and it requires less processing, it has been used for all the algorithms presented in

this thesis. A package or utility that implements these algorithms could easily have an option to transform the colourspace

8.4 Implementation Considerations

The algorithms presented here have been implemented as stand-alone C++ programs, using an image processing library to improve development speed. These programs are neither efficient in runtime and memory usage, nor particularly easy to use. This section considers the possibility of implementation of these algorithms in a commercial framework.

8.4.1 User Interfaces

A commercially viable system that generates alpha channels from input sequences is unlikely to be a stand–alone piece of software. Rather, it will be a plug–in or affiliate application to a larger compositing package such as Puffin Design's *Commotion* [10]. The suitability of this algorithm for incorporation into such a package has not been assessed.

Since there are bound to be cases where neither alpha estimation algorithms will achieve an acceptable result, tools must exist to allow an operator to modify the result. Many such tools already exist in compositing packages, and these algorithms must integrate with these tools.

8.4.2 Parallel Implementation

The hint image update algorithms were designed in such a way as to allow different computers to process multiple frames simultaneously. Only a small number of frames need be processed before processing can begin simultaneously on all the remaining frames. Further speed increases are possible by processing each frame using more than one computer. The simplest way of achieving this would be to split the images into multiple parts duplicating a small overlapping section in each parts. These part images are processed on different computers and then blended together to form a single image. If multi–processor computers are used, it could be possible to parallelise the algorithms on a finer level. Calculating the GMM for foreground and background clusters, for example, can be parallelised since they are independent of each other.

8.5 Critical Discussion

The algorithms presented here and previously published algorithms all operate (or appear to operate) solely by measuring differences between the colours of groups of pixels. All algorithms tend to fail in the same areas of an image, and often in a similar manner. This suggests that any superior colour–based algorithm will not perform significantly better than any of these. That is, to create a better alpha channel estimation, it will probably be necessary to turn to segmentation techniques other than colour.

However, these algorithms have performed remarkably well in many cases: Individual strands of hair are detected in the *Edith* image and the *Gema* image and the background is successfully rejected from the foreground (*i.e.* no hint of the original background is visible in the new foreground).

This work has involved the use of very high resolution images. In order to process a single image in reasonably time, each pixel must be processed very quickly. The Principal Axis and the Hint Update algorithms in particular operate extremely quickly, and are therefore suitable for use on high resolution images.

The lack of availability of high resolution sequences has made quantitive analysis of the quality of algorithms difficult. Ideally, more high resolution sequences should have been used in order to explore more situations under which the algorithms perform badly. Copyright and Intellectual Property agreements make it difficult to obtain image sequences. While the Computer Film Company kindly provided the *Rachael* image (which was recorded by them for in order to test their *Keylight* bluescreen package) film production companies prevented them and other post–production companies making available further sequences.

8.6 Recommendations for Future Work

8.6.1 Quantitive and Qualitive results

Assessing the performance of an alpha estimation algorithm is a difficult process. In order to compare more precisely the relative performance of difference algorithms, it would be necessary to develop more precise qualitive and quantitive measures of performance. A webpage including images produced by the different algorithms was prepared and sent to compositors

working in Motion Picture Post Production who were asked to assess them. However, there were not enough responses to draw any reasonable conclusions. The RMSE technique of producing quantitive results based on ground truth could be improved in order to obtain more precise information about the presence and size of errors and artifacts in the alpha channel. However, ground–truth for such images either does not exist or is at least as difficult (and maybe more time consuming) to generate than the estimates.

If better qualitive results were required, it would be possible to obtain such results by extending the samples presented in appendix C, including both single images and image sequences, and extending the sample from compositors to include film–goers.

8.6.2 Non Colour–based Segmentation

Since it is unlikely that further significant performance improvements can be made on algorithms that segment single images based on pixel colour alone, future work could look at other techniques. Using some model–based information (for example developing an algorithm that considers the shape of a head, the way light reflects and refracts in a strand of hair, or estimates the silhouette of a person wearing a black jumper in the frames where they are camouflaged in front of a black background) may help to segment in the difficult cases.

8.6.3 Combination of Algorithms

In many areas of the test images, the algorithms that run slowest (for example the Genetic Algorithm and that of Chuang *et al*) tended to perform better than the fastest ones, such as *Photoshop*. In the *Gema* image it can be seen that different algorithms perform best in certain areas of the image. Thus, speed and performance can be increased by selecting which algorithm to use for different areas of the image, and by combining the results of each algorithm together. Such a fusion of results would require an analysis step to examine the properties of an area in order to decide which algorithm would perform best, as well as a fitness assessment, so that a single result can be derived from the combination of different algorithms.

Figure 8.1: *Extended clean background generated from the* Dragonheart *sequence*

8.6.4 Extended Clean Backgrounds

Intuitively, the task of estimating for a pixel the clean foreground colour as well as its alpha value could be made simpler if one if the background was previously known. If the background is stationary, the background colour can be found by taking an image of the background before any foreground is present, or else pieced together by hand from different images where different parts of the background are visible. This is the basis behind the difference–based techniques described in section 4.2.

Chuang *et al* propose an adaptation to their algorithm to use such "extended clean backgrounds" in alpha estimation. However, their background must be pieced together by hand, or else photographed separately.

8.6.4.1 Generating extended clean backgrounds

Fig. 8.1 shows a partial background derived by combining the clean background from multiple images. This image was obtained by averaging together each estimate, favouring pixel values from images where the pixel is in the clean background area rather than estimated from the unknown area.

An attempt to produce a set of alpha channels for the image failed. The sequence has too much noise for the extended background to be a good estimate of the actual background colour and, since the background is relatively uniform, the standard techniques provide a reasonable estimate of the background colour without using this image.

Figure 8.2: *An attempt to generate a hint image automatically for frame 4 of the* Dragonheart *sequence, by marking as foreground all pixels that are detected to be moving by optical flow*

8.6.5 Hint Image Initialisation

8.6.5.1 Semi–automatic hint initialisation

The major part of user interaction is the drawing of a hint image. For this research, the image was drawn using the image editing package *Gimp*. A faster way of creating this image is desirable. The use of code similar to that by Tan and Ahuja *et al* [119], could be adopted. Alternatively, Intelligent Scissors [66] could be adapted to draw a line of variable width to mark the unknown area. The problem in either case is the decision of exactly how wide the unknown area should be. An option in *Photoshop*'s extract tool attempts to automatically change the width of the brush being used to draw the line (apparently by analysing the image gradient), but it is useful only in certain cases.

Making the drawing interactive — so that the alpha channel appears as the hint image is drawn — may make the task of initialising the algorithm much easier.

8.6.5.2 Automatic hint initialisation

Developing a system that would be able to segment any sequence of images without any form of human input is a complex problem. Fig. 8.2 shows an attempt at automatic hint image generation. The image was generated from the motion vector field. Any pixel that has a non–zero vector is considered to be moving. Assuming that the background is stationary and the entire foreground moving, this should detect the foreground. It fails partly because not all the

foreground is moving (the horse's back is relatively stationary) and partly because the smoothness constraint imposed in the optical flow has caused stationary parts of the background to be marked as moving.

Alternatively, hint images could be generated by attempting to derive an "extended clean background" image, which could be used to produce images for each frame using a difference image.

In the case of the segmentation of humans, model based techniques could be employed, that recognise the outline of a person, or the texture and colour of skin and cloth.

Removal of all human interaction would make this algorithm suitable for MPEG IV video compression, which supports semi-transparent layers (sprites) with alpha channels moving over each other.

8.7 Concluding Remarks

This thesis has presented a group of segmentation algorithms that operate on motion picture resolution images to produce alpha channels. The high resolution of these images lead to the development of a novel moving image segmentation algorithm, and fast and accurate approaches to alpha channel estimation of still images.

Although these algorithms seem to provide superior results to those previously published, they do not always produce acceptable results. In some cases, the new background over which the element is to be composited is such that imperfections in the alpha channel are not apparent. In other cases, alternative techniques (including manual painting) could be applied to improve the results. Even so, these algorithms may provide a useful contribution to motion picture special effects.

References

[1] R. Brinkmann, *The Art and Science of Digital Compositing*. Morgan Kaufmann, 1999.

[2] J. Cameron, *Titanic*. Twentieth Century Fox / Paramount Pictures, 1996.

[3] A. R. Smith and J. F. Blinn, "Blue screen matting," *Computer Graphics: Proceedings of the ACS*, pp. 259–268, 1996.

[4] A. R. Smith, "Image compositing fundamentals," Tech. Rep. 4.5, Microsoft, 15th August 1995.

[5] A. R. Smith, "Alpha and the history of digital compositing," Tech. Rep. 7.8, Microsoft, 15th August 1995.

[6] T. Porter and T. Duff, "Compositing digital images," *Computer Graphics: Proceedings of the ACS*, vol. 18, pp. 253–259, July 1984.

[7] D. E. Zongker, D. M. Werner, B. Curless, and D. H. Salesin, "Environment matting and compositing," in *Computer Graphics: Proceedings of SIGGRAPH '99*, pp. 205–214, 1999.

[8] P. Hanrahan and J. Lawson, "A language for shading and lighting calculations," *Computer Graphics: Proceedings of the ACS*, vol. 24, pp. 289–98, August 1994.

[9] Pixar, "RenderMan." http://www.pixar.com/.

[10] Puffin Designs, "Commotion." http://www.puffindesigns.com.

[11] Adobe Systems Inc., "Photoshop." http://www.adobe.com/products/photoshop.

[12] L. U. Borg and M. Hamburg, "Conversion of alpha-multiplied color data." United States patent number 6,208,351, March 2001.

[13] W. Skarbek and A. Koschan, "Colour image segmentation — a survey," tech. rep., University of Berlin, 1994.

[14] R. Hunt, *The Reproduction of Colour*. Fountain Press, 1995.

[15] A. Ford and A. Roberts, "Colour space conversions," tech. rep., University of Westminster, August 11 1998. Available online via http://www.inforamp.net/~ poynton/ColorFAQ.html.

[16] G. Healey, "Segmenting images using normalized color," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 22, pp. 64–73, Jan-Feb 1992.

[17] T. Gevers and A. Smeulders, "A comparative study of several color models for color image invariant retreival," in *Proc. 1st Int. Workshop on Image Databases and Multimedia Search*, pp. 17–23, 1996.

[18] J. Bradley, *XV version 3.10a: user manual*, 1994. pp. 100-101.

[19] J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes, and R. L. Phillips, *Introduction to Computer Graphics*. Addison-Wasley, 1993.

[20] M. Orchard and C. Bouman, "Color quantization of images," *IEEE Transactions on Signal Processing*, vol. 12, pp. 2677–90, December 1991.

[21] A. Bailey and A. Hooloway, *The Book of Colour Photography*. Mermaid Books, 1984.

[22] D. F. Malin, "A universe of color," *Scientific American*, vol. 269, pp. 56–61, August 1993.

[23] C. Woodworth, "How stuff works: How photographic film works." article online at http://www.howstuffworks.com/film.html, 2002.

[24] C. E. K. Mees, *Theory of the photographic process*. New York: Macmillan, 1942.

[25] S. Bernstein, *Film Production*. Focal Press, 1988.

[26] D. A. Spencer, *Colour Photography in Practice*. Sir Isaac Pitman and Sons ltd, 1939.

[27] Eastman Kodak Company, "Student filmmaker's handbook." http://www.kodak.com/country/US/en/motion/students/handbook/, September 2001.

[28] B. E. Bayer, "Color imaging array." United States patent number 3,971,065, July 20 1976.

[29] R. B. Merrill, "Color separation in an active pixel cell imaging array using a triple-well structure." United States patent number 5,965,975, Oct 12 1999.

[30] K. Findlater, D. Renshaw, J. Hurwitz, R. Henderson, T. Bailey, S. Smith, M. Purcell, and J. Raynor, "A cmos images sensor employing a double junction photodiode," in *IEEE Workshop on CCDs and Advanced Image Sensors*, pp. 60–63, June 2001.

[31] L. Joyeux, O. Buisson, B. Besserer, and S. Boukir, "Detection and removal of line scratches in motion picture films," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR*, vol. 1, pp. 548–53, 1999.

[32] M. B. o, A. Bertozzi, and G. Sapiro, "Navier-stokes, fluid dynamic, and image and video inpainting," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR*, vol. 1, pp. 355–62, 9th-14th December 2001.

[33] J. Chun Kit Yan and D. Hatzinakos, "Signal-dependent film grain noise removal and generation based on higher-order statistics," in *Proceedings of the IEEE Signal Processing Workshop on Higher-Order Statistics*, pp. 77–81, 1997.

[34] Cinesite Digital Film Center, "Greyscale transformations of cineon digital film data for display, conversion and film recording," Tech. Rep. 1.1, http://www.dotscw.com/doc/cineon2.pdf, April 12 1993.

[35] K. Hilton, "The end of the silver screen?," *Focus Magazine*, pp. 38–42, July 2001.

[36] G. Lucas, *Star Wars Episode II: Attack of the Clones*. Lucasfilm, 2002.

[37] Sony Corporation, *Sony Camcorder HDW-F900 user manual*, 1 ed., August 2000. obtained from http://www.cinematography.net/Files/F900_Ops.pdf.

[38] Arri Group, "Arriflex 435 advanced specifications." http://www.arri.com/prod/cam/435adv/435adv.htm.

[39] Panavision United Kingdom. Greenford, Middlesex, http://www.panavision.co.uk, April 2002.

[40] Foveon Inc, "F7 x3 cmos image sensor." http://www.foveon.com.

[41] G. A. Hance, S. E. Umbaugh, R. H. Moss, and W. V. Stoecker, "Unsupervised color image segmentation," *IEEE Engineering in Medicine and Biology*, pp. 104–110, January/February 1996.

[42] B. Sankur and M. Sezgin, "A survey of image thresholding techniques over categories," *Pattern Recognition*, 2001. Under Review. see http://www.busim.ee.boun.edu.tr/~sankur/SankurFolder/MVA_21.doc.

[43] Y. Mishima, "Soft edge chroma-key generation based on hexoctahedral color space." United States patent number 5,355,174, October 1994.

[44] Petro Vlahos, "Electronic composite photography with colour control." United States patent number 4,007,487, February 1977.

[45] Paul Vlahos, "Method and apparatus for adjusting parameters used by compositing devices." United States patent number 5,907,315, November 1996.

[46] CFC – The Computer Film Company, "Keylight." http://www.cfc.co.uk/cfc_website/site/% hspace0mmhtml/keylight.html.

[47] BBC Research, "The truematte system." http://www.bbc.co.uk/rd/projects/virtual/key-inh.html.

[48] reflecmedia, "Reflecmedia powerscreen." http://www.reflecmedia.com, April 2002.

[49] M. Ben-Ezra, "Segmentation with invisble keying signal," in *Proc. Computer Vision and Pattern Recognition, CVPR*, vol. 1, pp. 32–37, 2000.

[50] 3DV systems, "Zcam." http://www.3dvsystems.com.

[51] G. Yahav and G. Iddan, "Optical ranging camera." United States patent number 6,057,909, January 9 1997.

[52] L.-H. Chen and H.-Y. Liao, "Stereo correspondence by surface segmentation," *IEEE International Conference on Systems, man and Cybernetics*, vol. 1, pp. 593–98, 1992.

[53] M. Haindl and P. Zvid, "Fast segmentation of range images," in *Proceedings of the 9th International Conference in Image Processing, ICIAP '97*, vol. 1, pp. 295–302, September 1997.

[54] N. Ikonomatakis, K. N. Plataniotis, M. Zervakis, and A. N. Venetsanopoulos, "Region growing and region merging image segmentation," *13th International Conference on Digital Signal Processing Proceedings. DSP 1997*, vol. 1, no. 6, pp. 299–302, 1997.

[55] N. Ikonomakis, K. Plataniotis, and A. Venetsanopoulos, "A region-based color image segmentation scheme," in *Proceedings of the SPIE: Visual Communication and Image Processing '99*, vol. 3653, pp. 1202–1209, 1999.

[56] A. Siebert, "Dynamic region growing," in *Vision Interface '97*, 1997.

[57] R. Adams and L. Bischof, "Seeded region growing," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, pp. 641–48, june 1994.

[58] Hee Soo Yang and Sang Uk Lee, "Split and merge algorithm segmentation employing thresholding technique," in *International Conference on Image Processing*, vol. 1, pp. 239–42, 26–29 Oct 1997.

[59] Ye Xiu Qing, Huang Zhen Hua and Xiao Qiang, "Histogram based fuzzy c-mean algorithm for image segmentation," in *Proceedings, 11th IAPR Internaional Conference on Pattern Recognition. Conference C: Image, Speech and Signal Analysis*, vol. III, pp. 704–707, 1992.

[60] G. C. Karmaker and L. Dooley, "A generic fuzzy rule based technique for image segmentation," in *Proceedings International Conference on Acoustics, Speech and Signal Processing*, vol. 3, pp. 1577–80, 7-11 May 2001.

[61] N. Ray, J. Havlicek, S. T. Acton, and M. Pattichis, "Active contour segmentation guided by am-fm dominant component analysis," in *Proceedings of ISIP*, pp. 78–81, 2001.

[62] S. Lobregt and M. A. Viergever, "A discrete dynamic contour model," *IEEE transactions on Medical Images*, vol. 14, pp. 12–24, March 1995.

[63] G. Sapiro, "Note: Color snakes," *Computer Vision and Image Understanding*, vol. 68, pp. 247–253, november 1997.

[64] G. Sapiro, "Vector-valued active contours," in *Proceedings CVPR '96, IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pp. 680–685, 1996.

[65] X. Li and K. Wang, "Adaptive balloon models," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, pp. 434–439, 23-25 June 1999.

[66] E. N. Mortensen and W. A. Barrett, "Intelligent scissors for image composition," in *Proc. of the ACM SIGGRAPH 95: Computer Graphics and Interactive Techniques*, pp. 191–198, August 1995.

[67] E. N. Mortensen and W. A. Barrett, "Toboggan-based intelligent scissors with a four parameter-edge model," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR*, vol. 2, pp. 452–58, 1999.

[68] A. Trémeau and P. Colantoni, "Regions adjacency graph applied to colour image segmentation," *IEEE Transactions on Image Processing*, vol. 9, pp. 735–745, April 2000.

[69] E. N. Mortensen and W. A. Barrett, "A confidence measure for boundary detection and object selection," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR*, vol. 1, pp. 477–84, 9th-14th December 2001.

[70] L. Gool, P. Dewaele, and A. Oosterlink, "Texture analysis anno 1983," *Computer Vision, Graphics and Image Processing*, vol. 29, pp. 226–57, 1985.

[71] Corel Corporation, "Corel GALLERY 1,300,000." Software package, 1998.

[72] Y.-I. Ohta, T. Kanade, and T. Sakai, "Color information for region segmentation," *Computer Graphics and Image Processing*, vol. 13, pp. 222–41, 1980.

[73] T. Tan and J. Kittler, "Colour texture analysis using colour histograms," *IEE Proc. Vis. Image Signal Process*, vol. 141, pp. 403–412, December 1994.

[74] Yining Deng and B. Manjunath, "Unsupervised segmentation of color-texture regions in images and video," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, pp. 800–810, August 2001.

[75] Z. Gu, C. Duncan, E. Renshaw, M. Mugglestone, C. Cowan, and P. Grant, "Comparison of techniques for measuring cloud texture in remotely sensed meterological image data," *IEEE Proceedings Radar and Signal Processing*, vol. 136, pp. 236–248, October 1989.

[76] Y.-G. Wu, "Medical image compression by sampling dct coefficients," *IEEE Transactions on Information Technology in Biomedicine*, vol. 6, pp. 86–94, March 2002.

[77] R. Gonzales and R. Woods, *Digital Image Processing*. Addison–Wesley Publishing, 1993.

[78] J. Wei, "Image segmentation using situational DCT descriptors," in *Proceedings 2001 International Conference on Image Processing*, vol. 1, pp. 738–41, 7-10 October 2001.

[79] C. W. Therrien, *Decision Estimation and Classification*. John Wiley and Sons, 1989.

[80] D. P. Mital, "Texture segmentation using gabor filters," in *Fourth International Conference on knowledge-Based Intelligent Engineering Systems & Allied Technologies*, pp. 109–112, 30 Aug - 1 Sep 2000.

[81] M. Acharyya and M. K. Kundu, "Wavelet-based texture segmentation of remotely sensed images," in *Proceedings 11th International Conference on Image Analysis and Processing*, pp. 69–74, 2001.

[82] S.-C. Lam, "Texture feature extraction using gray level gradient based co-occurence matrices," in *IEEE Conference on Systems, Man, and Cybernetics*, vol. 1, pp. 267–71, 1996.

[83] C. C. Gotlieb and H. E. Kreszig, "Texture descriptors based on co-occurrence matrices," *Computer Vision, Graphics and Image Processing*, vol. 51, pp. 70–86, July 1990.

[84] S. Aksoy and R. Haralick, "Textural features for image database retrieval," in *Proceedings IEEE Workshop on Content-Based Access of Image and Video Libraries*, pp. 45–49, 1998.

[85] I. Haritaoglu, D. Harwood, and L. S. Davis, "w^4: Real-time surveillance of people and their activities," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 809–30, August 2000.

[86] Shen Yujian, He Xin, and Hao Zhihang, "Real-time temporal recursive filtering system for detection of small moving targets," in *Proceedings of the 5th International Conference on Signal Processing*, vol. 2, pp. 1045–48, 21-25 Aug 2000.

[87] M. Boninsegna and A. Bozzoli, "A tunable algorithm to update a reference image," *Signal Processing: Image Communication*, vol. 16, pp. 353–365, 2000.

[88] T. Horprasert, D. Harwood, and L. S. Davis, "A statistical approach for real-time robust background subtraction and shadow detection," in *Proceedings IEEE ICCV'99 FRAME-RATE Workshop*, September 1999.

[89] C. Lettera and L. Mastera, "Foreground / background segmentation in videotelephony," *Signal Processing: Image Communication*, vol. 1, pp. 182–89, October 1989.

[90] K.-W. Lee and J. Kim, "Moving object segmentation based on a statistical motion model," *Electronics Letters*, vol. 35, pp. 1719–20, 30th September 1999.

[91] T. Meier and K. N. Ngan, "Automatic segmentation of moving objects for video object plane generation," *IEEE Transactions Circuits Systesms and Video Technology (invited paper)*, vol. 8, no. 5, pp. 525–538, 1998.

[92] K. P. J Vass and X. Zhuang, "Automatic spatio-temporal video sequence segmentation," in *Proceedings of 1998 International Conference on Image Processing ICIP 1998*, vol. 1, pp. 958–62, 1998.

[93] R. Szeliski, "Image mosaicing for tele-reality applications," in *IEEE Workshop on Applications of Computer Vision (WACV '94)*, pp. 44–53, December 1994.

[94] J. R. Hidalgo and P. Salembier, "Robust segmentation and representation of foreground key–regions in video sequences," in *Proceedings International Conference on Acoustics, Speech and Signal Processing*, vol. 3, pp. 1565–68, 7-11 May 2001.

[95] M.-C. Lee, W.-g. Chen, C.-l. Bruce Lin, C. Gu, T. Makoc, S. I. Zabinsky, and R. I. Szeliski, "A layered video object coding system using sprite and affine motion model," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, pp. 130–45, February 1997.

[96] P. M. Antoszczyszyn, J. M. Hannah, and P. M. Grant, "Reliable tracking of facial feautures in semantic–based video coding," *IEE Proceedings Vision, Image and Signal Processing*, vol. 145, pp. 257–63, August 1998.

[97] T. Mitsunaga, Y. Yokoyama, and T. Totsuka, "Autokey: Human assisted key extraction," in *Computer Graphics: Proceedings of SIGGRAPH '95*, pp. 265–272, August 1995.

[98] A. M. Peacock, *Information Fusion for Improved Motion Estimation.* PhD thesis, University of Edinburgh Department of Electronics and Electrical Engineering, May 2001.

[99] B. Horn and B. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, pp. 185–203, 1981.

[100] F. D. Fabrice Moscheni and M. Kunt, "A new two–stage global/local motion estimation based on a background/foreground segmentation," in *International Conference on Acoustics, Speech and Signal Processing*, vol. 4, pp. 2261–2264, 1995.

[101] Y. Haung, K. Palaniappan, X. Zhaung, and J. E. Cavanaugh, "Optic flow field segmentation and motion estimation using a robust genetic partitioning algorithm," *IEEE Transactions and Pattern Analysis and Machine Intelligence*, vol. 17, pp. 1177—90, December 1995.

[102] S. Galić and S. Lončarić, "Spatio-temporal image segmentation using optical flow and clustering algorithm," in *First International Workshop in Image ans Signal Processing and Analysis*, pp. 63–68, June 14-15 2000.

[103] P. Erhan Eren, Y. Altunbasak, and A. Murat Tekalp, "Region–based affine motion segmentation using color information," in *1997 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP-97*, vol. 4, pp. 3005–09, 1997.

[104] J. Toro, F. Owens, and F. Medina, "Multiple motion estimation and segmentation in transparency," in *Proceedings IEEE International Conference on Acoutsics, Speech and Signal Processing, 2000. ICASSP '00*, vol. 4, pp. 2087–90, 5-9 June 2000.

[105] B. Bascle, P. Bouthemy, R. Deriche, and F. Meyer, "Tracking complex primitives in an image sequence," in *Proceedings of the 12th IAPR Conference on Pattern Recognition. Conference A: Computer Vision & Image Processing*, vol. 1, pp. 426–31, 9-13 October 1994.

[106] N. Peterfreund, "The velocity snakes: Deformable contour for tracking in spatio–velocty space," *Computer Vision and Image Understanding*, vol. 37, pp. 346–356, March 1999.

[107] L. Cohen and I. Cohen, "Finite–element methods for active contour models and balloons for 2d and 3d images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, pp. 1131–1147, November 1993.

[108] D. G. J Hall and G. Jones, "Segmenting film sequences using active surfaces," in *Proceedings, International Conference on Image Processing 1997*, vol. 1, pp. 751–754, 1997.

[109] A. Baumberg and D. Hogg, "An adaptive eigenshape model," in *British Machine Vision Conference* (D. Pycock, ed.), vol. 1, pp. 87–96, 1996.

[110] A. M. Peacock, "Hopalong — tracking and recognising pedestrians from gait." Honours Thesis, University of Edinburgh Departments of Artificial Intelligence and Computer Science, 1997/8.

[111] S. Wachter and H.-H. Nagel, "Tracking persons in monocular image sequences," in *Nonrigid and Articulated Motion Workshop*, pp. 2–9, 1997.

[112] Chuang Gu and Ming–Chieh Lee, "Semantic video object tracking using region-based classification," in *International Conference on Image Processing. ICIP 98*, vol. 3, pp. 643–47, 4-7 Oct 1998.

[113] Dongxiang Xu, Jenq-Neng Hwang, and Jun Yu, "An accurate region based object tracking for video sequences," in *IEEE 3rd Workshop on Multimedia Signal Processing*, pp. 271–76, 13-15 September 1999.

[114] M. Ruzon and C. Tomasi, "Alpha estimation in natural images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, vol. 1, pp. 18–25, June 2000.

[115] Y.-Y. Chuang, C. Brian, D. H. Salesin, and R. Szelsiki, "A bayesian approach to digital matting," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR*, vol. 2, pp. 264–71, 9th-14th December 2001.

[116] A. Berman, A. Dadourian, and P. Vlahos, "Method for removing from an image the background surrounding a selected object." United States patent number 6,134,346, October 17 2000.

[117] M. Evening, *Adobe Photoshop 6.0 for Photographers*. Focal Press, 2001.

[118] C. Gu and M. C. Lee, "Semiautomatic segmentation and tracking of semantic video objects," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, pp. 572–84, September 1998.

[119] K.-H. Tan and N. Ahuja, "Selecting objects with freehand sketches," in *Proceedings Eighth IEEE Internaional Conference on Computer Vision*, vol. 1, pp. 337–44, 7-14 July 2001.

[120] K.-H. Tan and N. Ahuja, "A representation of image structure and its application to object selection using freehand sketches," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR*, vol. 2, pp. 677–83, 9th-14th December 2001.

[121] P. Mattis and S. Kimball, "GIMP - The Gnu Image Manipulation Package." http://www.gimp.org.

[122] L. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, pp. 338–353, 1965.

[123] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Application*. Springer–Verlag, 2 ed., 2000.

[124] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. B. Flannery, *Numerical Recipes in C: the art of scientific computing*. Cambridge University Press, 2 ed., 1992.

[125] K. P. Fishkin and B. A. Barsky, "A family of new algorithms for soft filling," *Computer Graphics*, vol. 18, pp. 235–41, July 1984.

[126] F. Perparata and S. Hong, "Convex hulls of finite sets of points in two and three dimensions," *Communications of the ACM*, vol. 20, pp. 87–93, February 1977.

[127] A. Berman, A. Dadourian, and P. Vlahos, "Comprehensive method for removing from an image the background surrounding a selected object." United States patent number 6,134,345, October 17 2000.

[128] E. Dijsktra, "A note on two problems in conexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–71, 1959.

[129] A. Gibbons, *Algorithmic Graph Theory*. Cambridge University Press, 1985.

[130] D. E. Goldberg, *Genetic Algorithms in search, optimization, and machine learning*. Addison-Wesley, 1989.

[131] R. Thomson and T. Arslan, "An evolutionary algorithm for the multi-objective optimisation of primitive operator filters," in *Proceedings of the 2002 Congress on Evolutionary Computation*, vol. 1, pp. 37–42, 12–17 May 2002.

[132] M. Raymer, L. Kuhn, and W. Punch, "Knowledge diesecovery in biological datasets usina a hybrid bayes classifier/evolutionary algorithm," in *Proceedings of the IEEE 2nd International Symposium on Bioinformatics and Bioengineering Conference*, pp. 236–245, 2001.

[133] M. J. Black, *Robust Incremental Optical Flow*. PhD thesis, Yale University Department of Computer Science, 1992. YALEU-DCS-RR-923.

[134] M. J. Black and P. Anandan, "A framework for the robust estimation of optical flow," in *Proceedings, Fourth International Conference on Computer Vision*, pp. 231–36, 11-14 May 1993.

[135] M. J. Black, "Optical flow software." http://www.cs.brown.edu/people/black/code.html, April 2002.

[136] P. Hillman, J. Hannah, and D. Renshaw, "Alpha estimation in high resolution images and image sequences," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR*, vol. 1, pp. 1063–68, 9th-14th December 2001.

[137] P. Salembier, A. Oliveras, and L. Garrido, "Anti-extensive connected operators for image and sequence processing," *IEEE Transactions on Image Processing*, vol. 7, pp. 555–70, April 1998.

[138] D. Straus, "Linux helps bring titanic to life," *Linux Journal*, February 1998.

[139] R. Cohen, *Dragonheart*. Universal Pictures, 1996.

[140] A. M. Peacock, "Vision systems code documentation," tech. rep., University of Edinburgh Department of Electronics and Electrical Engineering, June 16 2000.

Appendix A
Test Data

Since the standard images often used to demonstrate image processing algorithms are not suitable for alpha channel generation, a new set of test images have been used. The sizes and sources of these images is listed in table A.1.

These images were obtained from a variety of sources. The *Rachael* test image was kindly provided by the Computer Film Company in Cineon format. A small utility was written which converted this to linear format for processing. The *Gema* image was scanned from a 35mm slide, and the *Edith* image obtained from the Corel imaging bureau. High resolution image sequences were difficult to obtain. Copyright restrictions made it impossible to obtain image sequences scanned directly from first generation motion picture stock. The resolution of standard digital camcorders is not high enough to simulate motion picture sequences. Instead, a digital stills camera was used to create the *Teddy* sequence. In order to test the hint image update algorithms on an image sequence originated on motion picture film, a 35mm trailer print from the film *Dragonheart* [139] was scanned using a conventional slide scanner. The resultant images suffered from misalignment, and required manual correction to align them as accurately as possible. The scanned print suffers from dust and scratches, as well as excessive film grain noise. 35mm trailers are usually prints taken from a second or third generation master. Thus, they contain film grain noise from three or four individual pieces of film. A scan of the original negative film would be much less noisy.

Image	Resolution	Source	Lighting	Digitisation
Gema (fig. A.2)	1612 × 1673 8 bpc	35mm slide (Fuji Velvia 100 ASA)	Daylight: Afternoon sunlight in top left	Nikon LS-2000 slide scanner: TIFF image
Rachael (fig. A.1)	640 × 480 14 bpc	Unknown tungsten balanced film	Tungsten lamps on foreground and on bluescreen backing	CFC film scanner. Provided in Cineon format. Converted to linear for processing
Teddy (fig. A.3,A.4)	1546×1155×3 8bpc	Olympus single CCD Digital camera (gain set to 100 ASA)	Single tungsten lamp at left	Direct 8 bit TIFF output
Edith (fig. A.5)	2240 × 2204 8bpc	Unknown (Corel Stock Image number 678058)	Unknown (possibly multiple flash)	Unknown: PCD format
Dragonheart (fig. A.6,A.7)	2286 × 1224 × 21 8bpc	Motion picture film	Unknown (backlit with sunlight)	Motion picture trailer print scanned on Nikon LS-2000 Slide scanner. Sequence contains high film grain noise, dust and scratches. Scanned Images aligned by hand.

Table A.1: *Details of test images*

Figure A.1: *The* Rachael *test image*

Figure A.2: *The* Gema *test image*

Figure A.3: *The* Teddy *test image - frame 1 of the* Teddy *Sequence*

(a) Frame 2

(b) Frame 3

Figure A.4: *Frames 2 and 3 of the* Teddy *Sequence*

Figure A.5: *The* Edith *test image*

Figure A.6: *Frame 15 of the* Dragonheart *sequence*

(a) Frame 1 **(b)** Frame 2 **(c)** Frame 3

(d) Frame 4 **(e)** Frame 5 **(f)** Frame 6

(g) Frame 7 **(h)** Frame 8 **(i)** Frame 9

(j) Frame 10 **(k)** Frame 11 **(l)** Frame 12

(m) Frame 13 **(n)** Frame 14 **(o)** Frame 15

(p) Frame 16 **(q)** Frame 17 **(r)** Frame 18

(s) Frame 19 **(t)** Frame 20 **(u)** Frame 21

Figure A.7: *The* Dragonheart *sequence*

Appendix B
Publication

- P. Hillman, J. Hannah, and D. Renshaw, "Alpha estimation in high resolution images and image sequences," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR*, vol. 1, pp. 1063–68, 9th-14th December 2001, Kauai Hawaii USA.

Appendix C
Webpage for evaluation

The following webpage was sent to various compositors working in the Motion Picture Industry, who were asked to assess the performance of six different algorithms.

Alpha Channel Estimation in High Resolution Images

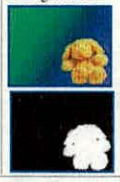
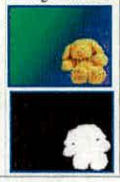
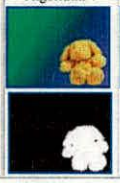
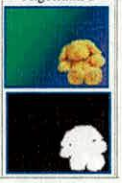
From this page you can see results of six algorithms running on four different test images. Click any image to see the **much** larger results. Datasizes quoted are for the colour images (in TIFF format). Mattes are gifs and are significantly smaller.

The thumbnails for the composites and alpha channels are all the same, and are not scaled versions of the actual results.

You will probably have to **save the images to file** in order to view them properly

You can download all the images in one go if you prefer: pmhresults.zip (84MB). Since this is a compressed file, you'll save quite a few MBs of download!

Click here for an out-of-date website explaining an old version of my algorithm, where you can also download a paper published at CVPR 2001

Image	Algorithm 1	Algorithm 2	Algorithm 3	Algorithm 4	Algorithm 5	Algorithm 6
Rachael 640 x 480 (1.1MB)	Algorithm 1 	Algorithm 2 	Algorithm 3 	Algorithm 4 	Algorithm 5 	Algorithm 6 
Gema 1612 x 1673 (10.29MB)	Algorithm 1 	Algorithm 2 	Algorithm 3 	Algorithm 4 	Algorithm 5 	Algorithm 6 
Edith 2240 x 2024 (12.97MB)	Algorithm 1 *No results available*	Algorithm 2 	Algorithm 3 	Algorithm 4 	Algorithm 5 	Algorithm 6 
Teddy 1546x1155 (6.81MB)	Algorithm 1 	Algorithm 2 	Algorithm 3 	Algorithm 4 	Algorithm 5 	Algorithm 6 

Peter M Hillman
Last modified: Mon Jul 8 19:23:14 BST 2002

Appendix D
Example code

D.1 Coding Style

The algorithms implemented in this thesis were implemented as stand–alone C++ programs. This approach allowed advantages such as platform independency, batch processing capability and access to debugging and profiling tools. Code was written to run on both Sun workstations and Linux machines. Hint images were created using *Gimp* [121] which, along with the *xv* image viewing package, was used to view the output images of these programs. *Gimp* was also used to create some of the compose images used in this thesis. The compose program listed in Section D.3.2 was used for the *Dragonheart* sequence.

D.1.1 Program Size

Obviously a considerable amount of software was written in order to produce the results presented in this thesis. For example, the Principal Axis and hint update algorithms are approx 2500 lines of code in length. Since differing algorithms share many similarities, an object orientated approach was adopted so that individual parts could be re-used easily.

D.2 The Vision Systems Group Library

The Vision Systems Group Library [140] was used to provide basic image handling operations. This is a C++ library consisting of basic image operations (such as reading from disk and access to individual pixels channels) as well as operations such as image rotation, edge detection. Since the library did not contain basic colour operations, a `Pixel` object was contributed to the library. This provides code for colourspace transforms as well as pixel arithmetic, such as multiplication of pixels by scalar values.

The header file for the floating point version of this class is reproduced in Section D.3.1 to illustrate the coding style adopted.

D.3 Sample Code

D.3.1 VS_floatpixel Class

```
//
// file      : VS_floatpixel.h
// author    : Peter M Hillman (pmh)

//
// (c) 2000 Vision Systems (ISG), the University of Edinburgh
//

#ifndef __VS_FLOATPIXEL_H__
#define __VS_FLOATPIXEL_H__

//
// enumerated colour model type
//
// non : no specific color model
// rgb : red, green, blue
// uvw : U'V'W' (linear space model)
// hsv : Hue (0-360) Sat (0-100) Value (0-100)
// XYZ : CIE XYZ (0-MaxIntensity)
// Lab : CIE L*a*b* space
// hsi : HSI space
// err : An error occurred.

////////////////////////////////
// class  VS_floatpixel
//
// Class for storing pixels with up to 3
// channels and floating point
// channel intensities, with primitive
// color transforms and pixel math
////////////////////////////////


class VS_floatpixel
{

public:

    //
    // type of model in pixel
    //
    enum VS_pixel_model
    {non=0,rgb,uvw,hsv,xyz,Lab,hsi,err};


protected :

    int nModel;

    double dMaxInten;  // max intensity

public :

    // channel values (can be accessed directly)

    double pdChannel[3];
```

```
//
// Constructor
//
VS_floatpixel(void);

//
// change a normal pixel into a floating point one
//
VS_floatpixel(VS_pixel & normpix);

//
// to convert a floatpixel back to a VS_pixel
// to allow write-to-file
//

VS_pixel Fixed(void);

//
// operations on pixels
//

// threshold values to range 0 -- dMaxInten
int clip(void);

// return magnitude of pixel (Euclidean)
double mag(void);

// Euclidean distance between pixels
double distance(VS_floatpixel p);

//
// +/-/* operate on each channel
// separately (vector/scalar arithmetic)
//

VS_floatpixel operator+= (const double);
VS_floatpixel operator-= (const double);
VS_floatpixel operator*= (const double);

VS_floatpixel operator+ (double);
VS_floatpixel operator- (double);
VS_floatpixel operator* (double);

// vector/vector arithmetic

VS_floatpixel operator+= (const VS_floatpixel);
VS_floatpixel operator-= (const VS_floatpixel);
VS_floatpixel operator+ (const VS_floatpixel);
VS_floatpixel operator- (const VS_floatpixel);


// tidy channel access
double & operator[] (int d)
{ return pdChannel[d];}

int GetModel(void) const { return nModel;}

//
// Read pixel from frame (return success)
//

int GetPixel(const VS_frame & frame,int nX,int nY);
int GetPixel(const VS_frame_interp & frame,double dX,double dY);
```

```
    //
    // maximum intensity of pixel;
    //
    double MaxInten(void) const { return dMaxInten; }
    void MaxInten(double d) {if (d>0.0) dMaxInten=d;}

    //
    //
    // Transform colour model
    // to given colour model
    // (return old colour model or error)
    //

    int Transform(int);

    int FakeTransform(int newmodel)
        { int y=nModel;nModel=newmodel; return y;}

    //
    // print out pixel values.
    //

    friend ostream & operator <<
       (ostream &os,VS_floatpixel & p)
    {
        return os << p.pdChannel[0]
                  << ' ' << p.pdChannel[1]
                  << ' ' << p.pdChannel[2];
    }
};
#endif
```

D.3.2 Compose Program

```
#include <VS_floatpixel.h>
#include <VS_frame_io.h>

int main(int argc,char * argv[])
{
    //
    // check correct arguments
    //

    if(argc!=5)
    {
        cerr << "usage: " << argv[0] <<
             " background alpha foreground output\n";
    }

    //
    // read images
    //

    VS_frame back=VS_frame_io().LoadFrame(argv[1]);
    VS_frame alpha=VS_frame_io().LoadFrame(argv[2]);
    VS_frame foreground=VS_frame_io().LoadFrame(argv[3]);

    //
    // process each pixel in foreground image
    //
```

```
    for( int x=0 ; x<foreground.GetWidth() ; x++ )
    {
        for( int y = 0 ; y < foreground.GetHeight() ; y++ )
        {

            VS_floatpixel b,f,o;

             //
             // read new background and foreground pixels
             //

            b.GetPixel( back , x , y );
            f.GetPixel( foreground , x , y );

             //
             // read value from alpha channel and
             // scale to [0..1]
             //

            float a=((float) alpha.GetIntensity(x,y))/
                            alpha.MaxIntensity();

             //
             // find output pixel colour
             //

            o=f*a + b*(1.0-a);

             //
             // write pixel back to image
             //

            o.Fixed().SetPixel(&foreground,x,y);

        }
    }

    //
    // save file
    //

    VS_frame_io t;
    t.DisplayFrame(foreground,argv[4]);
}
```